Sentaurus™ Visual User Guide

Version N-2017.09, September 2017

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

About This Guide

The Synopsys Sentaurus™ Visual tool is part of Sentaurus Workbench Visualization. It is a plotting software for visualizing data from simulations and experiments. Sentaurus Visual enables users to work interactively with data using both a graphical user interface and a scripting language for automated tasks.

Related Publications

For additional information, see:

- The TCAD Sentaurus release notes, available on the Synopsys SolvNet® support site (see Accessing SolvNet on page xviii).
- Documentation available on SolvNet at https://solvnet.synopsys.com/DocsOnWeb.

Conventions

The following conventions are used in Synopsys documentation.

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

Customer Support

Customer support is available through the Synopsys SolvNet customer support website and by contacting the Synopsys support center.

Accessing SolvNet

The SolvNet 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 SolvNet site:

1. Go to the web page at https://solvnet.synopsys.com.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register.)

If you need help using the site, click **Help** on the menu bar.

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

Contacting Your Local TCAD Support Team Directly

Send an e-mail 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.

Free and Open Source Software Licensing Information

This document contains licensing information relating to Synopsys's use of free and open-source software with or within the Sentaurus Visual software. This document identifies the free and open-source packages used, the licenses that Synopsys understands such packages to be available under, and, where available, a link to the package licensor's or distributor's website or the website where such package was downloaded from. The copyright notices and licensing information from the packages, as provided in the packages, have also been reproduced below. While Synopsys has sought to provide complete and accurate licensing information for each of the free and open-source packages identified herein, Synopsys does not represent or warrant that the licensing information provided herein is correct or error-free. Recipients of the Sentaurus Visual software should investigate the identified packages and sources to confirm that accuracy of the licensing information. Recipients are also encouraged to notify Synopsys of any inaccurate information or errors found in this document.

****
Boost 1.59.0
License: Boost Software License, version 1.0
Source/Licensor: http://www.boost.org/
****

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

****
CPPTCL 1.1.3
License: Modified Boost Library
Source/Licensor: http://cpptcl.sourceforge.net/
****

Each source file contains the following notice:

```
//
// Copyright (C) 2004-2006, Maciej Sobczak
//
// Permission to copy, use, modify, sell and distribute this software
// is granted provided this copyright notice appears in all copies.
// This software is provided "as is" without express or implied
// warranty, and with no claim as to its suitability for any purpose.
//
```

The above is the shortest license I have found.

I have copied this wording from one of the source files of the Boost library.

Just to explain the above terms:

1. The library is free (like in "freedom").
2. You can use it for any purpose you wish. You can even sell it.
3. You can modify the source files. If you do this, please feel free to add your own copyright notices wherever you want, if you want. You may also add more restrictive license terms. You only have to keep the above notice unchanged in all source files.
4. You can combine this library or its modified version with other software. The license claims nothing that would affect the resulting work. See point 3. if you need to make any modifications.
5. You can compile this library or its modified version to the binary form. The license does not affect the compiled result in any way. In particular, you do not need to (but you still can) display the above copyright notice to the final user, and you do not need to (but you still can) inform the final user that your product was prepared with the help of this library.
6. Do not sue me for my (and your) mistakes and errors.

As you see, this license is very permissive.

****
DejaVu Fonts 2.33
License: Subject to multiple licenses
Source/Licensor: http://dejavu-fonts.org/
****

Fonts are (c) Bitstream (see below). DejaVu changes are in public domain.
Glyphs imported from Arev fonts are (c) Tavmjong Bah (see below)

Bitstream Vera Fonts Copyright

Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved. Bitstream Vera is a trademark of Bitstream, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Bitstream" or the word "Vera".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Bitstream Vera" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the names of Gnome, the Gnome Foundation, and Bitstream Inc., shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from the Gnome Foundation or Bitstream Inc., respectively. For further information, contact: fonts at gnome dot org.

Arev Fonts Copyright

Copyright (c) 2006 by Tavmjong Bah. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the modifications to the Bitstream Vera Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Tavmjong Bah" or the word "Arev".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Tavmjong Bah Arev" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL TAVMJONG BAH BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the name of Tavmjong Bah shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from Tavmjong Bah. For further information, contact: tavmjong@free. fr.

$Id: LICENSE 2133 2007-11-28 02:46:28Z lechimp $

****
ImageMagick 6.7.4
License: ImageMagick License
Source/Licensor: http://imagemagick.com/script/index.php
****

Before we get to the text of the license, lets just review what the license says in simple terms:

It allows you to:

* freely download and use ImageMagick software, in whole or in part, for personal, company internal, or commercial purposes;

* use ImageMagick software in packages or distributions that you create;

* link against a library under a different license;

* link code under a different license against a library under this license;

* merge code into a work under a different license;

* extend patent grants to any code using code under this license;

* and extend patent protection.

It forbids you to:

* redistribute any piece of ImageMagick-originated software without proper attribution;

* use any marks owned by ImageMagick Studio LLC in any way that might state or imply that ImageMagick Studio LLC endorses your distribution;

* use any marks owned by ImageMagick Studio LLC in any way that might state or imply that you created the ImageMagick software in question.

It requires you to:

* include a copy of the license in any redistribution you may make that includes ImageMagick software;

* provide clear attribution to ImageMagick Studio LLC for any distributions that include ImageMagick software.

It does not require you to:

* include the source of the ImageMagick software itself, or of any modifications you may have made to it, in any redistribution you may assemble that includes it;

* submit changes that you make to the software back to the ImageMagick Studio LLC (though such feedback is encouraged).

A few other clarifications include:

* ImageMagick is freely available without charge;

* you may include ImageMagick on a DVD as long as you comply with the terms of the license;

* you can give modified code away for free or sell it under the terms of the ImageMagick license or distribute the result under a different license, but you need to acknowledge the use of the ImageMagick software;

* the license is compatible with the GPL V3.

* when exporting the ImageMagick software, review its export classification.

Terms and Conditions for Use, Reproduction, and Distribution

The legally binding and authoritative terms and conditions for use, reproduction, and distribution of ImageMagick follow:

Copyright 1999-2013 ImageMagick Studio LLC, a non-profit organization dedicated to making software imaging solutions freely available.

1. Definitions.

License shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

Licensor shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

Legal Entity shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, control means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

You (or Your) shall mean an individual or Legal Entity exercising permissions granted by this License.

Source form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

Object form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

Work shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

Derivative Works shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship.

For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

Contribution shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner.

For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as Not a Contribution.

Contributor shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted.

If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

* You must give any other recipients of the Work or Derivative Works a copy of this License; and

* You must cause any modified files to carry prominent notices stating that You changed the files; and

* You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

* If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places:

within a NOTICE text file distributed as part of the Derivative Works;

within the Source form or documentation, if provided along with the Derivative Works; or,

within a display generated by the Derivative Works, if and wherever such third-party notices normally appear.

The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an AS IS BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act

only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

How to Apply the License to your Work

To apply the ImageMagick License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information (don't include the brackets). The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

> Copyright [yyyy] [name of copyright owner]
>
> Licensed under the ImageMagick License (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at
>
> http://www.imagemagick.org/script/license.php
>
> Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

****
Mesa 17.0.0 and X11 drivers
License: MIT
Source/Licensor: http://www.mesa3d.org/
****

Disclaimer

Mesa is a 3-D graphics library with an API which is very similar to that of OpenGL.* To the extent that Mesa utilizes the OpenGL command syntax or state machine, it is being used with authorization from Silicon Graphics, Inc.(SGI). However, the author does not possess an OpenGL license from SGI, and makes no claim that Mesa is in any way a compatible replacement for OpenGL or associated with SGI. Those who want a licensed implementation of OpenGL should contact a licensed vendor. Please do not refer to the library as MesaGL (for legal reasons). It's just Mesa or The Mesa 3-D graphics library.

* OpenGL is a trademark of Silicon Graphics Incorporated.

License / Copyright Information

The Mesa distribution consists of several components. Different copyrights and licenses apply to different components. For example, the GLX client code uses the SGI Free Software License B, and some of the Mesa device drivers are copyrighted by their authors. See below for a list of Mesa's main components and the license for each. The core Mesa library is licensed according to the terms of the MIT license. This allows integration with the XFree86, Xorg and DRI projects.

The default Mesa license is as follows:

Copyright (C) 1999-2007 Brian Paul All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

****
muParser 2.2.5
License: MIT
Source/Licensor: http://muparser.beltoforion.de/
****

Copyright (c) 2011 Ingo Berg

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

****
SGI GLU Library and Ext. headers (glext.h) (glxext.h)
License: SGI Free B
Source/Licensor: http://oss.sgi.com
****

SGI FREE SOFTWARE LICENSE B (Version 2.0, Sept. 18, 2008)

Copyright (C) [dates of first publication] Silicon Graphics, Inc. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice including the dates of first publication and either this permission notice or a reference to http://oss.sgi.com/projects/FreeB/ shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL SILICON GRAPHICS, INC. BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of Silicon Graphics, Inc. shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from Silicon Graphics, Inc.

****
VTK 7.1
License: BSD
Source/Licensor: http://www.vtk.org/
****

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither name of Ken Martin, Will Schroeder, or Bill Lorensen nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 1 Getting Started

This chapter presents basic information about starting Sentaurus Visual.

Introducing Sentaurus Visual

Sentaurus Visual allows you to visualize complex simulation results generated by physical simulation tools in one, two and three dimensions. You can visualize data for an initial understanding and analysis, and then modify the plots to gain a new perspective.

Sentaurus Visual can be used to create plots that display fields, geometries, and regions, including results such as p-n junctions and depletion layers. It also allows you to view I–V curves and doping profiles, and provides tools to zoom, pan, and rotate images. You also can extract data using measure and probe tools.

The graphical user interface (GUI) provides direct and easy-to-use functionality, as well as advanced controls for expert users. With the Sentaurus Visual GUI, you can systematically visualize devices as xy, 2D, and 3D plots.

Starting Sentaurus Visual

Sentaurus Visual can be started either from the command line or from Sentaurus Workbench.

From Command Line

To start Sentaurus Visual from the command line, type:

```
svisual
```

The following example loads the dataset associated with a file and generates its plot:

```
svisual n2_fps.tdr
```

Command-Line Options

When starting from the command line, the following options (which can be obtained by typing `svisual -h`) can be used:

```
Usage: svisual [options] [FILES]

Description:
   Sentaurus Visual is a tool to display and analyze structures and curves.

Options:
   -h[elp]             : Display this help message.
   -v[ersion]          : Print the Sentaurus Visual version.
   -m[esa]             : Force run Sentaurus Visual with Mesa.
   -glx                : Force run Sentaurus Visual with GLX driver if it
                         exists.
   -avx                : Force run Sentaurus Visual with AVX support, if it is
                         available (Mesa must be enabled).
   -b[atch]            : Run in pure batch mode (requires a script file).
   -batchx | -bx       : Run in batch mode that allows picture exporting
                         (virtual X server, requires a script file).
   -s[cript]           : Force to execute the next files as Sentaurus Visual
                         scripts.
   -i[nspect] | -f     : Force to execute the next files as Inspect scripts.
   -library_path       : Look for Tcl library files at the next path.
   -nolibrary          : Disable Tcl library auto-loading.
   -nowait             : Do not wait for license to become available.
   -verbose            : Log every Sentaurus Visual Tcl command executed to the
                         log file.
   -slowscript | -ss   : Redraw plots automatically after each command;
                         execution of script is slower.
   -geoms              : Load only geometries from the next list.
```

NOTE Sentaurus Visual can run solely in batch mode, that is, no display is required and scripts can be run using a shell. This mode is fast but has some disadvantages, for example, exporting graphics only works in the GUI mode. To overcome this, use the `-batchx` option.

From Sentaurus Workbench

Sentaurus Visual is integrated in Sentaurus Workbench. You can start Sentaurus Visual by either:

- Clicking a node, which displays the Node Explorer. In the Node Explorer, in the **Viewer** box, select **svisual** and click the **Launch** button next to it.
- Clicking the Visualize toolbar button and selecting **Sentaurus Visual**.

Sentaurus Visual can receive node data and can be inserted into tool flows.

NOTE Sentaurus Visual can run in batch mode (`-b` option), which is especially useful when used within tool flows. In this context, the use of macro files is also of interest (see Chapter 6 on page 139).

Accelerating the Rendering of Graphics

In Sentaurus Visual, 2D and 3D plots are rendered using OpenGL acceleration, which can produce significant differences in performance depending on the configuration of the machine where Sentaurus Visual runs.

By default, Sentaurus Visual always runs in the best supported graphics mode it can find, using the graphics card of the machine to render plots. If there is not a compliant renderer, Sentaurus Visual reverts to a generic Mesa driver for graphics rendering.

If your computer has a graphics card but Sentaurus Visual runs with Mesa rendering, you can force Sentaurus Visual to run with a GLX driver using the `-glx` option. If a GLX driver cannot be found, Sentaurus Visual exits with an error message. To force Mesa rendering, use the `-mesa` option.

NOTE You cannot use both the `-glx` option and the `-mesa` option simultaneously.

CHAPTER 2 Graphical User Interface

This chapter describes the graphical user interface of Sentaurus Visual.

Introducing Sentaurus Visual GUI

The graphical user interface (GUI) of Sentaurus Visual has different areas (see Figure 1). The selection and properties panels are located to the left of the main window, the plot area displays the different visualizations, the Tcl Command panel is in the lower part, and the toolbars are located on the sides of the main window.

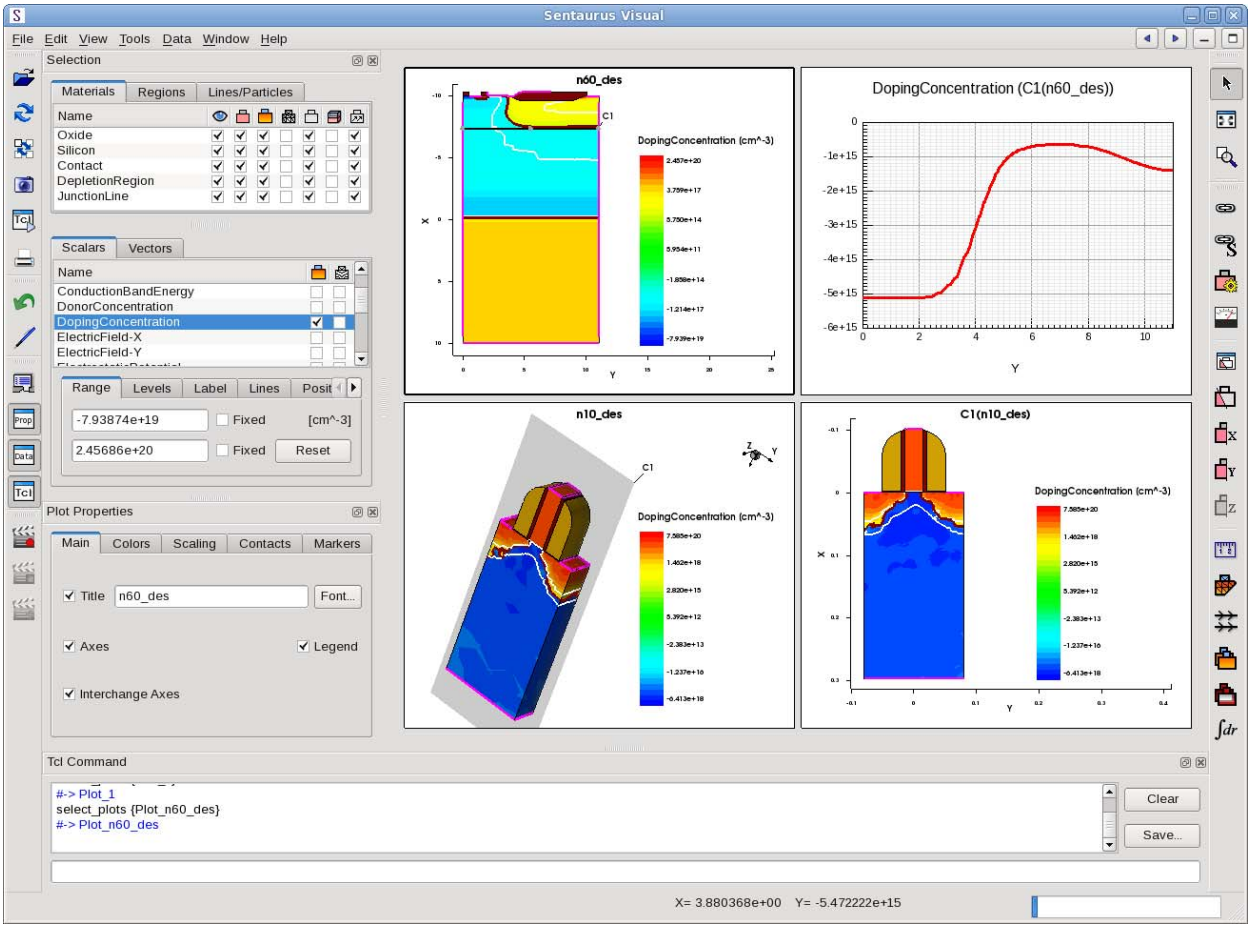


Figure 1 Main window of Sentaurus Visual showing different plots in the plot area

For detailed information about the menus and toolbars, see Appendix B on page 303.

NOTE You can customize the Sentaurus Visual GUI. Different options are available, for example, you can detach the panels, adjust their size, and move the toolbars to another part of the main window.

Menu Bar

The menu bar allows you to access the main operations of Sentaurus Visual such as opening files, showing and hiding toolbars, configuring Sentaurus Visual, manipulating loaded data, and organizing plots in the plot area. Table 1 lists the menus that are available in the GUI.

Table 1 Menus

Menu	Description
File	Loads plots and scripts, reloads data, and exports and prints plots.
Edit	Selects plots, and selects settings for Sentaurus Visual.
View	Shows and hides toolbars and panels; plot settings and performance options.
Tools	Accesses analysis tools.
Data	Views loaded datasets, and deletes selected plots.
Window	Organizes and manages active plots.
Help	Provides information about Sentaurus Visual.

Toolbars

Toolbars offer quick access to commonly used functions that are also available from the different menus (see Toolbars on page 310).

Table 2 Toolbars

Toolbar	Description
File	Loads plots and scripts, reloads data, and exports and prints plots.
Edit	Undoes operations, and displays toolbar for drawing shapes and inserting text onto plots.
View	Accesses zoom operations and subsampling.
Tools	Accesses analysis tools.

Table 2 Toolbars

Toolbar	Description
Custom Buttons	Accesses custom buttons. See Creating Custom Buttons to Access Scripts on page 15.
Movies	Records animated images.
Look	Shows or hides panels.

NOTE One toolbar is always visible to allow you to show and hide the Tcl Command panel and to organize the data selection and properties panels into tabs.

Plot Area

The plot area displays the active plots. The toolbars and panels change depending on the type of plot that is selected.

Tcl Command Panel

The Tcl Command panel shows valuable information about the commands used to manipulate and display data in Sentaurus Visual. It can be used to enter commands manually, which is very helpful when running complex calculations on datasets and displaying results.

The Tcl Command panel has three main areas:

- The main pane shows every action performed in Sentaurus Visual since the session started.
- On the right side, the **Clear** button is used to delete the command history from the main pane, and the **Save** button is used to store everything that was executed into a script file, so that it can be run without repeating all the operations.
- In the lower pane, you can manually enter Tcl commands.

For a more detailed explanation of Tcl commands and scripting, see Chapter 6 on page 139 and Appendix A on page 143.

Selection and Property Panels

Two panels are located by default at the left side of the main window:

- The Selection panel displays the data to visualize and shows which data is already displayed. This panel is different for xy plots, and 2D and 3D plots. The differences are explained in Chapter 4 on page 37 and Chapter 5 on page 59.
- The Property panel lists the selected object properties. By default, it shows the plot properties. It will change after an object is selected. This panel may also be displayed (if it is arranged behind the Selection panel or if it is hidden) at its last position in the main window if you double-click the object.

If the plot area is empty (no plots are created or all plots are hidden), these panels are always hidden. After a plot is created, by default, both panels open even if both were closed by the user in the last session.

To change this default behavior:

1. Choose **Edit > User Preferences**.
2. In the User Preferences dialog box, expand **Application > Common**.
3. Under Force Panels to Show, clear one or both of the **Selection Panel** option and **Properties Panel** option.
4. Click **Save**.

Quick Access to Tabs of Plot Properties and Axis Properties Panels

NOTE The quick access operation for axes applies to xy and 2D plots only.

The Plot Properties panel or the Axis Properties panel *must be already open* for these quick access operations to work. To open the Plot Properties panel, double-click an empty part of the plot if another panel is active. To open the Axis Properties panel, double-click any axis in the plot area.

You can quickly access different tabs of the Plot Properties panel or the Axis Properties panel by clicking particular parts of a plot in the plot area:

- Click the plot title to display the **Main** tab of the Plot Properties panel.
- Click an axis title (for example, X) to display the **Title/Scale** tab of the Axis Properties panel.

- Click any tick label on an axis (for example, -5) to display the **Ticks** tab of the Axis Properties panel. This operation applies only to 2D plots.
- Click an axis line to display the **Main** tab of the Axis Properties panel.

CHAPTER 3 Basic Operations

This chapter describes the basic operations that are common to all types of plot in Sentaurus Visual.

Loading Files

You can load files from the GUI or the command line, for example:

```
svisual [file1.tdr file2.tdr ...]
```

To load a file from the GUI:

1. Choose **File** > **Open**.
2. In the dialog box that is displayed, browse to the file you want to open, or type the file name in the **File name** field.
3. Click **Open**.

An opened file consists of datasets. A dataset is a structure containing data that is plotted on xy, 2D, or 3D space. For example, a `.plt`, or `.plx` file can consist of one or more datasets, and `.tdr` files usually consist of only one dataset.

NOTE To select multiple files, hold the Ctrl key when you click the required files to load.

Supported File Formats

Sentaurus Visual supports the most commonly used file formats, including: `.csv`, `.plt`, `.plx`, `.tdr`, and `.tif`.

For more information about the TDR format, see the *Sentaurus™ Data Explorer User Guide*.

Loading Scripts

Sentaurus Visual can load scripts from the command line. For example, you can simply type `svisual` with the path of the Tcl (`.tcl`) script, and Sentaurus Visual automatically detects the script.

Tcl scripts with the file extension `.tcl` run native Sentaurus Visual commands, while Inspect scripts need the `-inspect` or `-f` option to run Inspect commands in compatibility mode.

Most Inspect commands are fully supported, although some commands have only partial support and some commands are not supported at all. For detailed information about support for Inspect libraries and commands, see Appendix D on page 323. For detailed information about Inspect commands, refer to the *Inspect User Guide*.

From the GUI, choose **File** > **Run Tcl Script** to display a dialog box where you can select the script to load. The scripts loaded using the GUI run native commands only.

Reloading Plot Files

Sometimes, there are changes to the datasets from outside Sentaurus Visual. These changes can be shown without closing Sentaurus Visual.

To reload a specific dataset:

- Choose **File** > **Reload Selected** or press Shift+F5.

To reload all datasets:

- Choose **File** > **Reload All** or press the F5 key.

NOTE Not all changes to a dataset can be reloaded. For example, if the original structure was two dimensional, the reloaded data is expected to belong to a 2D plot. If, after changes, the dataset now contains data for a 3D structure, Sentaurus Visual cannot reload this plot.

Managing Loaded Information

Sentaurus Visual provides a dialog box to manage the information loaded in the current session.

Choose **Data** > **View Info Loaded** to display the Manage Loaded Data dialog box (see Figure 2) with all the data that is currently active and the option of removing plots and datasets.

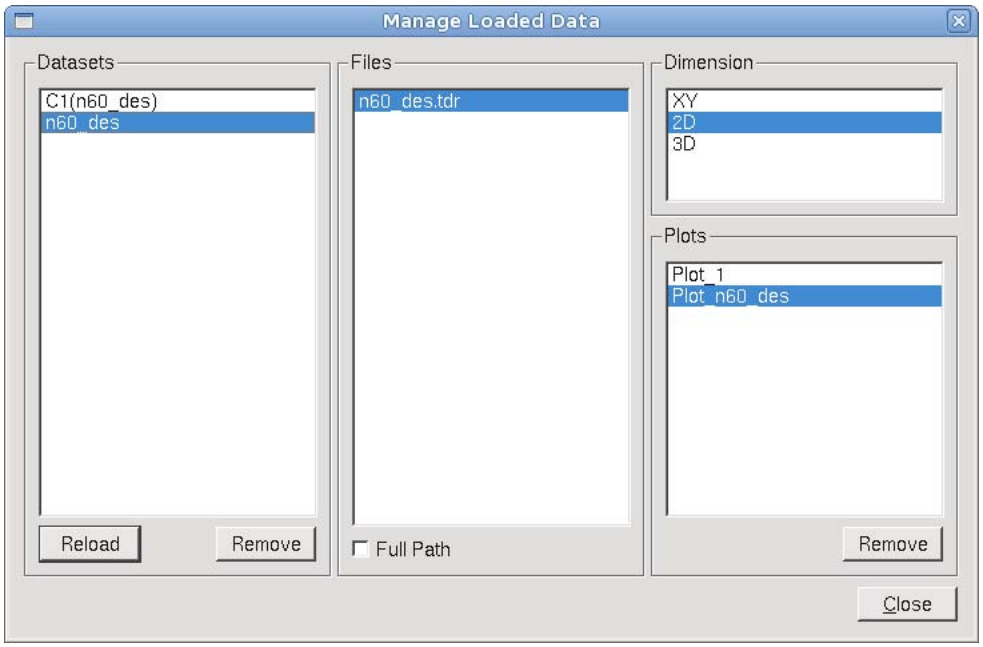


Figure 2 Manage Loaded Data dialog box showing active data

To delete all xy plots:

1. In the Dimension pane, click **XY**.
2. In the Datasets group box, click the **Remove** button.

To delete a plot:

1. In the Plots pane, click the plot to be deleted.
2. Under the Plots pane, click the **Remove** button.

NOTE Deleting a plot does not delete the datasets associated with it. However, deleting a dataset removes the associated 2D or 3D plots. For xy datasets, only the curves that use the datasets are deleted.

Customizing Settings

You can customize the settings of Sentaurus Visual using the User Preferences dialog box (see Figure 3).

To display the User Preferences dialog box, choose **Edit** > **Preferences**.

You can also import or export settings to a file by clicking the **Import** button or the **Export** button. To restore the preferences to their defaults, click the **Reset** button.

Alternatively, you can import and export settings using Tcl commands:

- To import previously saved settings, use the `import_settings` command (see import_settings on page 213).
- To export the current settings, use the `export_settings` command (see export_settings on page 177).

NOTE Settings are applied the next time you launch Sentaurus Visual.

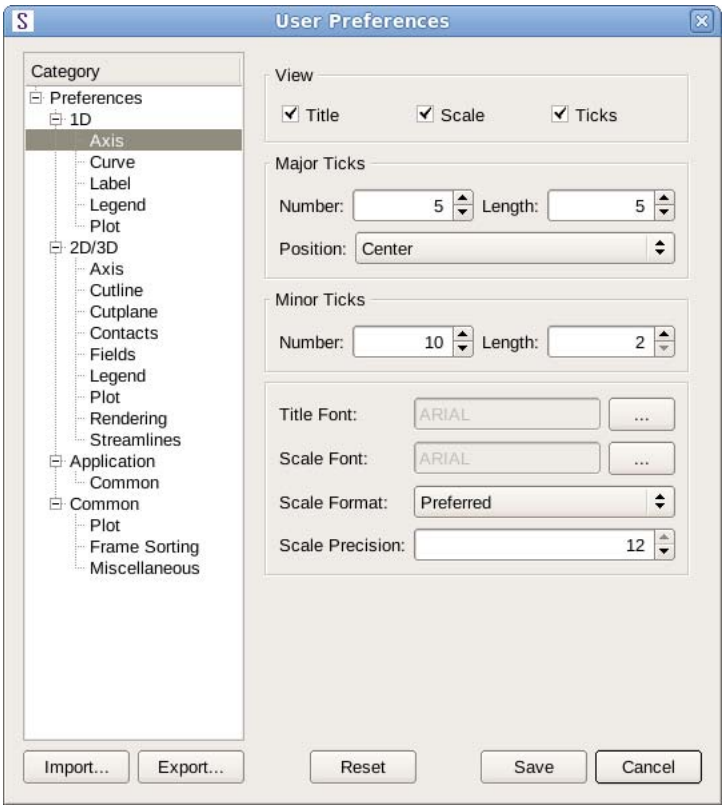


Figure 3 User Preferences dialog box showing selected settings

On Linux operating systems, user preferences are stored in the following file:

```
~/.config/Synopsys/SVisual.conf
```

Creating Custom Buttons to Access Scripts

You can create buttons to make it easier to execute or load Tcl script files. Custom buttons are added to the Custom Buttons toolbar, which is located immediately below the menu bar and, by default, has the **+** and **-** buttons.

Each new button can be set up to load a Tcl script file or to execute directly a Tcl script code. For each button, you can assign text or an icon to display as the button, as well as a tooltip.

Custom buttons can be loaded at the start of a Sentaurus Visual session when it loads scripts stored in the Tcl script library (see Script Library on page 142).

When you click the button, Sentaurus Visual displays a message before and after the script is executed, so you can identify the section of the commands that is executed using the button. The message identifies the button that has been clicked by its name and description.

For more information, see add_custom_button on page 147, get_input_data on page 194, list_custom_buttons on page 219, and remove_custom_buttons on page 246.

Working With Plots

Sentaurus Visual offers different modes when interacting with plots. These modes are independent for each plot instance and are enabled using toolbar buttons. However, you can apply mode changes to a group of plots by selecting the plots *before* applying the mode change.

Modes When Interacting With Plots

By default, a group of linked plots shares the same mode. This behavior can be switched off in the User Preferences dialog box (expand **Common** > **Miscellaneous**) by clearing the **Plot Mode** option (see Figure 4 on page 16). If this option is not selected, you can use special linking for a specific group of linked plots to change this behavior (see Linking Plots on page 18).

The modes temporarily modify the behavior of the left mouse button, allowing you to perform specific operations.

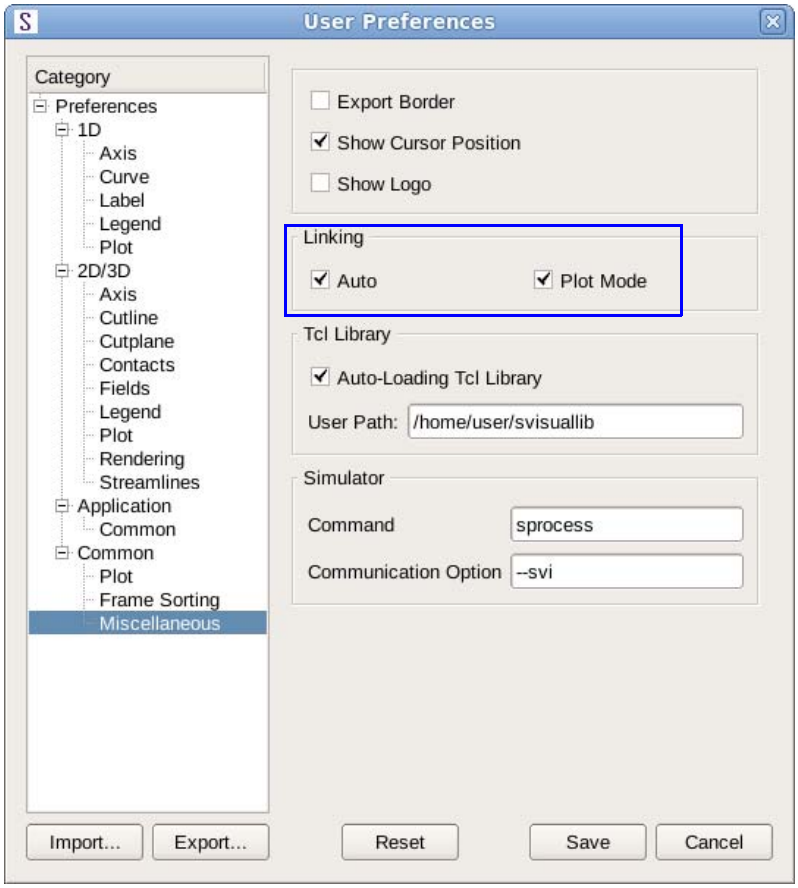


Figure 4 User Preferences dialog box showing Plot Mode option selected (the default)

Common Modes

All modes are enabled by clicking a toolbar button. The current mode remains active until you select another mode:

- The Selection mode (the default mode) allows you to select and move all objects inside plots (such as curves, legend, rectangles, and ellipses).
- The Zoom mode allows you to drag the left mouse button to draw a box. When you release the mouse button, the area delimited by the box will be magnified.
- The Probe mode allows you to extract data by clicking in the plot. For xy plots, curve data is extracted (see Probe Tool on page 54). For 2D and 3D plots, structure data is extracted (see Probe Tool on page 96).

XY Plot–Only Modes

All modes are enabled by clicking a toolbar button. The Drawing mode displays a submenu of drawing options:

- The Draw Line mode allows you to draw a line with the left mouse button.
- The Draw Rectangle mode allows you to draw a rectangle with the left mouse button.
- The Draw Ellipse mode allows you to draw an ellipse with the left mouse button.
- The Insert Text mode allows you to insert a text box with the left mouse button at a specified position.

NOTE For all of these drawing options, when you release the mouse button, the current mode finishes and it changes to the Selection mode.

2D Plot–Only Modes

All modes are enabled by clicking a toolbar button. The modes specific to 2D plots only are:

- The Cut X mode, the Cut Y mode, and the Cut Z mode allow you to generate an axis-aligned (x, y, or z) cutline at a specified position (see Cutlines in 2D Plots on page 113). When you release the mouse button, the current mode finishes and it changes to the Selection mode.
- The Cutline mode allows you to draw a cutline with the left mouse button. When you release the mouse button, the current mode finishes and it changes to the Selection mode.
- The Ruler mode allows you to draw a line with the left mouse button to perform a measurement. If you hold the Ctrl key while you click the mouse button, the snap-to-mesh mode is enabled (see Measuring Distances on page 94). This mode remains active until you select another mode.
- The Drawing mode displays a submenu of drawing options:
  - The Draw Line mode allows you to draw a line with the left mouse button.
  - The Draw Rectangle mode allows you to draw a rectangle with the left mouse button.
  - The Insert Text mode allows you to insert a text box with an arrow that can be repositioned using the left mouse button.

NOTE For all of these drawing options, when you release the mouse button, the current mode finishes and it changes to the Selection mode.

3D Plot–Only Modes

All modes are enabled by clicking a toolbar button. The modes specific to 3D plots only are:

- The Spherical Rotation mode allows you to perform a rotation in spherical coordinates using the left mouse button. This mode overrides the Selection mode as the default mode and remains active until you select another mode.
- The Rotation Axis X mode, the Rotation Axis Y mode, and the Rotation Axis Z mode allow you to perform a rotation around a fixed x-axis, y-axis, or z-axis with the left mouse button. This mode cannot select plot elements (aside from the legend) and remains active until you select another mode.
- The Cut X mode X, the Cut Y mode Y, and the Cut Z mode Z allow you to generate an axis-aligned (x, y, or z) cutplane at a specified position (see Cutplanes in 3D Plots on page 123). When you release the mouse button, the current mode finishes and it changes to the Selection mode or the Spherical Rotation mode (depending on which mode was last active).
- The Ruler mode allows you to draw a line with the left mouse button to perform a measurement. If you hold the Ctrl key while you click the mouse button, the snap-to-mesh mode is enabled (see Measuring Distances on page 94). This mode remains active until you select another mode.

Linking Plots

The feature of linking plots can be used to compare two similar models, as it allows you to manipulate elements from one plot of the group, and the linked elements will change on all plots of the group. Elements that can be linked include material/region selection, field selection and properties, movement and rotation, cutplanes and cutlines, axes properties (only in xy plots and 2D plots), legend properties, curves properties, grid properties, and plot properties.

To link plots:

1. Select the plots to be linked by holding the Shift key and clicking the required plots.
2. Click the toolbar button.

The linking operation links all properties except for y-axes and y2-axes in xy plots and streamlines in 2D and 3D plots. For customized linking properties, special linking can be used to link only specified properties and to set the remaining properties individually.

Plot linking also links the plot mode. This behavior is switched on by default and can be changed in the User Preferences dialog box (expand **Common** > **Miscellaneous**) by clearing the **Plot Mode** option (see Figure 4 on page 16). However, special linking can be used to change this behavior for particular groups of linked plots.

NOTE All plot properties are linked by default, including the properties of the plot title, except the text of the title, which is independent of the other plots regardless of which linking option is selected.

To use special linking:

1. Select the plots to be linked by holding the Shift key and clicking the required plots.
2. Click the toolbar button.

The properties that can be linked or unlinked with special linking include:

- Common properties:
  - Legend settings and movement
  - Plot properties and plot mode
- Only for xy plots:
  - Curve settings and grid settings
  - Axis settings (divided into x-, y-, and y2-settings)
- Only for 2D plots:
  - Axis properties
- Only for 2D and 3D plots:
  - Material or region selection
  - Field selection and field properties
  - Deformation and streamlines
  - Cuts

Undoing Operations

Most user interaction commands in Sentaurus Visual have undo functionality that allows you to revert recent changes to the visualization.

To undo an operation, click the toolbar button or use the `undo` Tcl command (see undo on page 299).

Displaying Multiple Plots

In the plot area, multiple plots can be displayed in a grid, with or without keeping the aspect ratio. In addition, you can arrange plots horizontally or vertically.

Grid Orientation

Plots can be displayed in a grid configuration (see Figure 5) by choosing **Window > Tile Grid**.

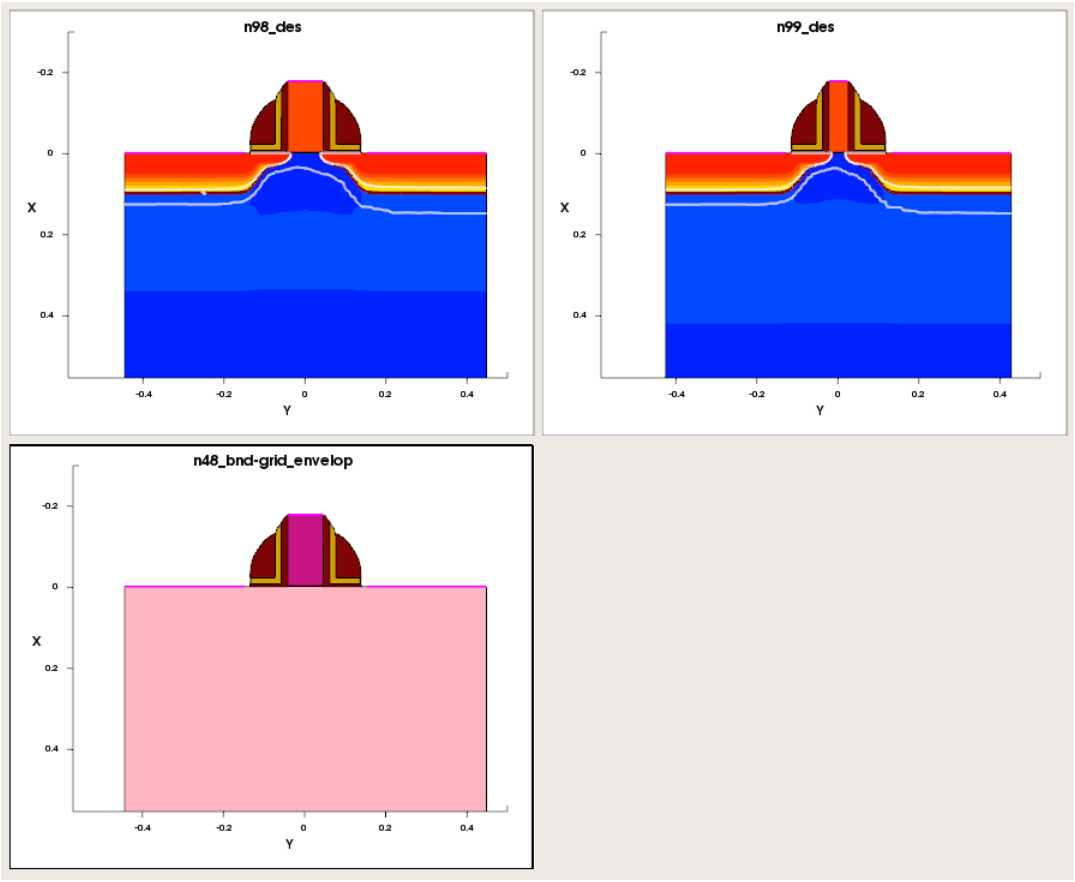


Figure 5 Multiple plots keeping the same aspect ratio

By default, the aspect ratio between plots is preserved, but this can be changed by clearing the **Keep Aspect Ratio** option of the Manage Frames dialog box (see Figure 9 on page 23). Figure 6 on page 21 shows plots where the aspect ratio is not maintained.

When the aspect ratio is not maintained, the unused space is filled with the last plot frame, but the aspect ratio of the structure is preserved.

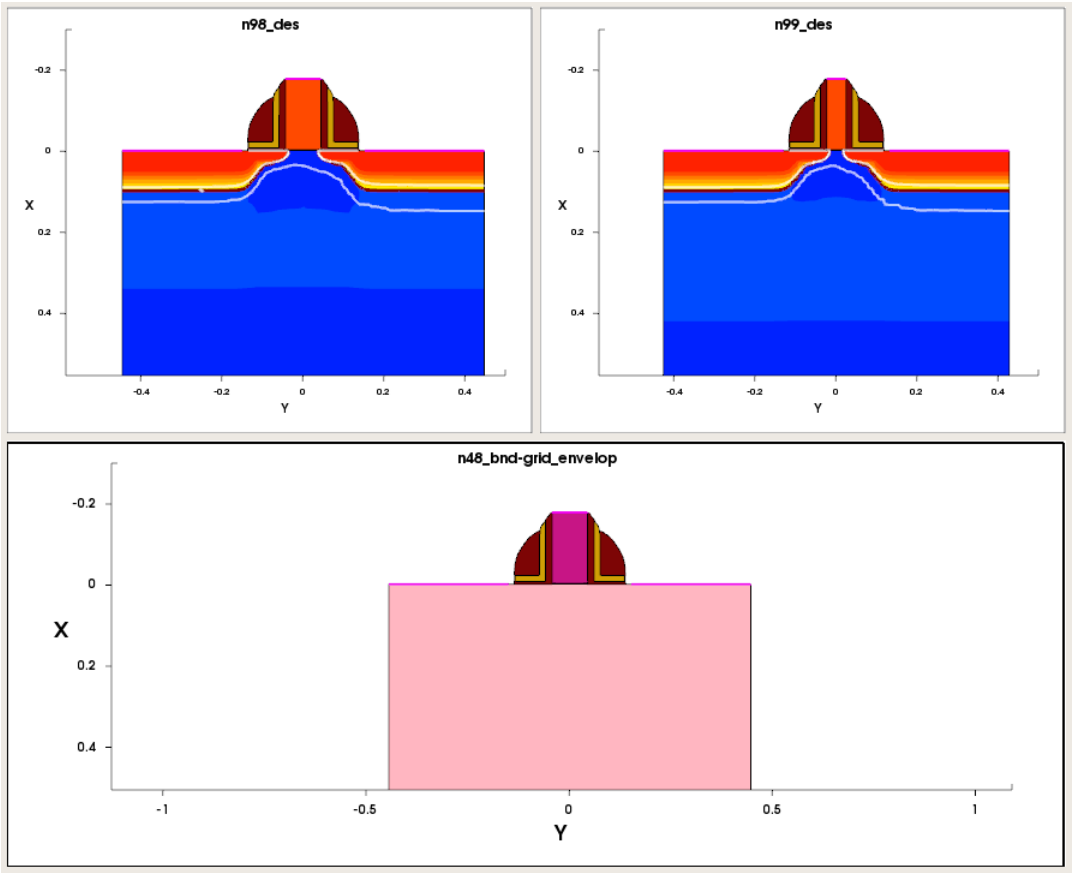


Figure 6 Multiple plots without keeping the same aspect ratio

Vertical Orientation

Plots can be arranged vertically (see Figure 7) by choosing **Window** > **Tile Vertically**.

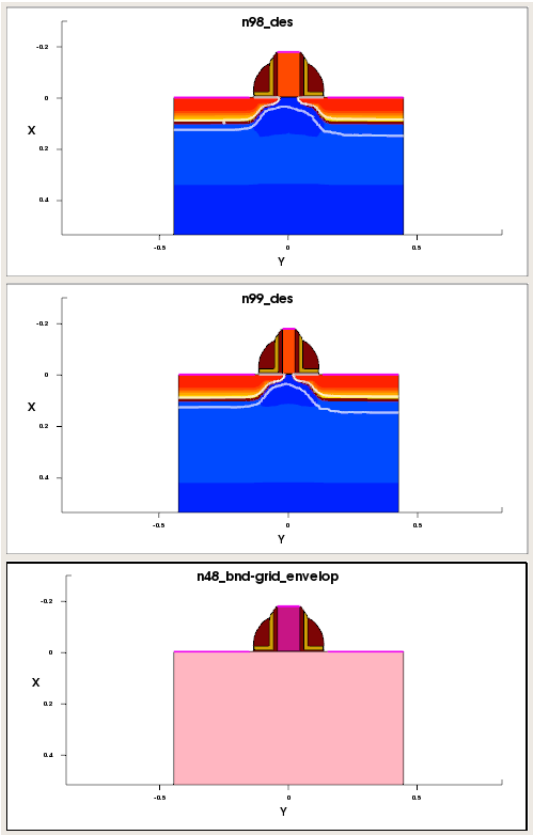


Figure 7 Plots arranged vertically

Horizontal Orientation

Plots can be arranged horizontally (see Figure 8) by choosing **Window** > **Tile Horizontally**.

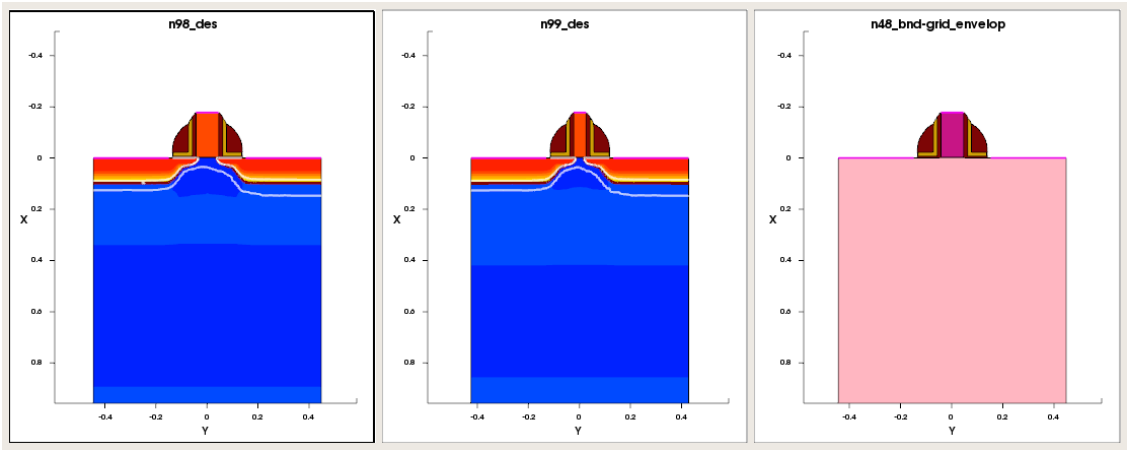


Figure 8 Plots arranged horizontally

Managing Frames

More advanced sorting options can be configured in the Manage Frames dialog box (see Figure 9). To display the dialog box, choose **Window** > **Manage Frames**.

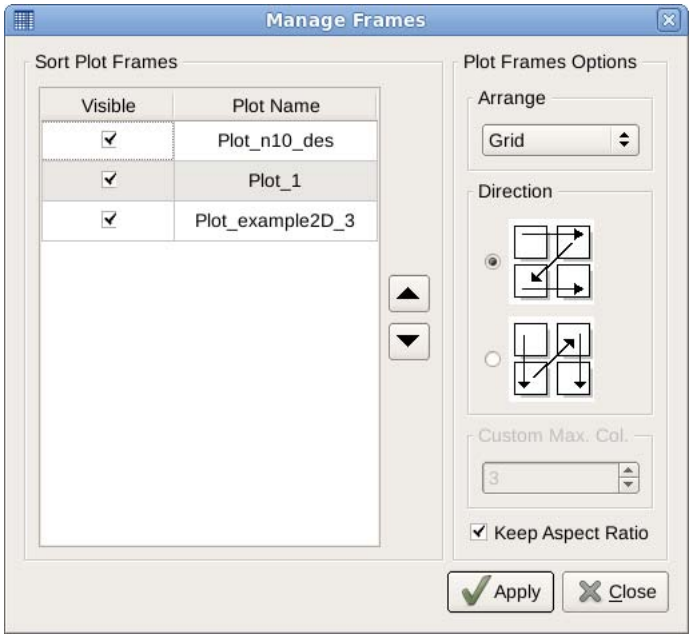


Figure 9 Manage Frames dialog box

Features available include setting a custom grid, sorting plots in the plot area, and changing the direction in which new plots are placed on the grid. In addition, you can manage plots by minimizing them or restoring them using the **Visible** option.

Drawing Inside Plots

Sentaurus Visual allows you to draw inside plots and to insert labels to allow plot customization. This feature is not available for 3D plots.

Inserting Text

To insert text inside a plot, click the toolbar button, or use the `draw_textbox` command (see draw_textbox on page 173).

The text properties such as font size, font color, and position can be changed using the `set_textbox_prop` command (see set_textbox_prop on page 290). This feature is available for xy and 2D plots.

There is a difference between the behavior of text boxes in xy plots and 2D plots, which is related to the coordinate system:

- In xy plots, the lower-left corner of the text box is placed at a specified point using the world coordinate system {x, y}. The text box keeps its position even if you perform a panning operation.

  The *world coordinate system* is a Cartesian coordinate system where the positions of objects, such as curves and drawn objects, are defined. The scale of the axes shows the world coordinate values.
- In 2D plots, the anchor arrow also exhibits this behavior, so it is placed at a specified point using the world coordinate system. However, the text box in 2D plots always retains its specified visual position even if you perform a panning operation. This is because the text box uses relative normalized screen coordinates, so its position is defined by numbers from 0.0 to 1.0, that is, {0.0, 0.0} for the lower-left corner of the plot area and {1.0, 1.0} for the upper-right corner of the plot area.

Drawing Rectangles

NOTE This feature is available only for xy and 2D plots.

To draw a rectangle, click the toolbar button or use the `draw_rectangle` command (see draw_rectangle on page 172).

You can edit a rectangle using the user interface or the `set_rectangle_prop` command (see set_rectangle_prop on page 283).

To delete a rectangle, select the rectangle and press the Delete key.

To delete multiple rectangles simultaneously, use the `remove_rectangles` command (see remove_rectangles on page 251).

To list the rectangles inside a plot, use the `list_rectangles` command (see list_rectangles on page 230).

Drawing Ellipses

NOTE This feature is available only for xy plots.

To draw an ellipse, click the toolbar button or use the `draw_ellipse` Tcl command (see draw_ellipse on page 170).

You can edit an ellipse using the user interface or the `set_ellipse_prop` command (see set_ellipse_prop on page 271).

To delete an ellipse, select the ellipse and press the Delete key.

To delete multiple ellipses simultaneously, use the `remove_ellipses` command (see remove_ellipses on page 249).

To list the ellipses inside a plot, use the `list_ellipses` command (see list_ellipses on page 223).

Drawing Lines

NOTE This feature is available only for xy and 2D plots.

To draw a line, click the toolbar button or use the `draw_line` Tcl command (see draw_line on page 171).

You can edit a line using the user interface or the `set_line_prop` command (see set_line_prop on page 277).

To delete a line, select the line and press the Delete key.

To delete multiple lines simultaneously, use the `remove_lines` command (see remove_lines on page 250).

To list the lines inside a plot, use the `list_lines` command (see list_lines on page 226).

Exporting Plots

You can export plots to an image file. Sentaurus Visual supports exporting plots to the following file formats: BMP, EPS, JPG, JPEG, PNG, PPM, TIF, TIFF, XBM, and XPM.

To export plots:

1. Choose **File** > **Export Plot**, press Ctrl+E, or click the toolbar button.

   The Export Plot dialog box is displayed (see Figure 10 on page 27).
2. Select the option to export multiple plots.
3. Select the option for the resolution.
4. Click **OK**.

NOTE The **User Defined** option is not available if you select the **All Plots to One File** option.

NOTE If you specify a custom resolution rather than select the **Screen Resolution** option, the exported plots might look different on-screen due to rescaling to the chosen resolution.

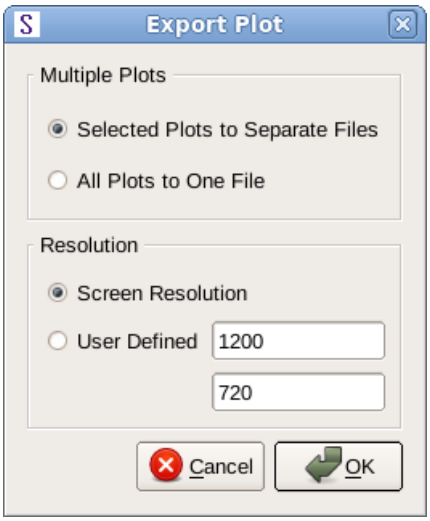


Figure 10 Export Plot dialog box

Exporting Movies

You can export several captures of one or more plots to generate an animated GIF file.

Starting a New Movie

To start a new movie:

1. Choose **Tools** > **Movies** > **Start Recording**, or click the toolbar button.

   The Start Recording dialog box is displayed, where you can generate frames for the movie (see Figure 11 on page 28).

2. Select the resolution:

   - The **Screen Resolution** option keeps the size of the current view.
   - The **User Defined** option allows you to specify the size of the capture in pixels.

3. Click **OK**.

Figure 11 Start Recording dialog box

Adding Frames in a Movie

To add a new frame in the movie:

1. Click the required plot to select it. To select multiple plots, hold the Shift key while clicking the plots.
2. Choose **Tools** > **Movies** > **Add Frames**, or click the toolbar button.

If several plots are selected, one frame for each plot is generated.

Exporting a Movie

To export a movie:

1. Choose **Tools** > **Movies** > **Stop Recording**, or click the toolbar button.

   The Export Movie dialog box is displayed (see Figure 12 on page 29).
2. To see a preview of a frame, click an item in the left pane.
3. Select the frames to export from the left pane.

   To make multiple selections, drag to highlight the frames or hold the Ctrl key while clicking the frames. At least one frame must be selected.
4. If required, change the order of the frames by selecting a frame from the left pane, and clicking the **Up** or **Down** button.

   NOTE The movie is recorded sequentially from the first frame to the last frame.

5. Set the duration of each frame in the **Frame Duration** field (the unit is 1/100 s).
6. Click **OK** to save the file.

   Clicking **Cancel** will delete the entire frame buffer.
7. In the dialog box that is displayed, ensure that the file has the `.gif` extension. Add the extension if it is missing.

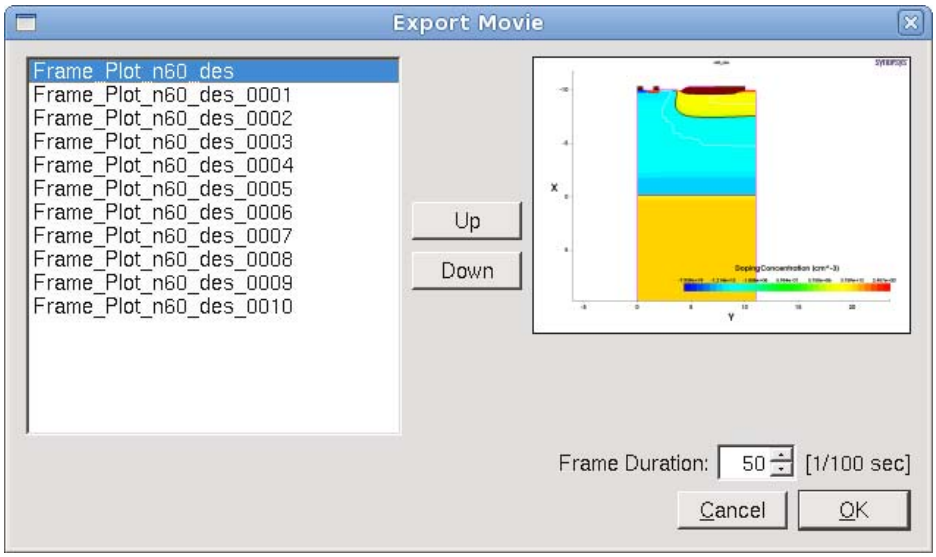


Figure 12 Export Movie dialog box

Printing Plots

You can print selected plots by either clicking the toolbar button, or choosing **File > Print Plots**, or pressing Ctrl+P.

The Printer dialog box is displayed, where you can select a printer and set print properties.

NOTE All plots are printed on one page.

Zooming and Panning

To take a closer look at significant details on a plot, there are various ways to zoom. On a selected plot, you can zoom by using the mouse wheel, or by clicking the middle mouse button and moving to the top or bottom of the screen. To pan, drag while holding the right mouse button.

Zoom Tool

The zoom tool is used to magnify a particular area of a plot.

To select the zoom tool:

1. Click the toolbar button.
2. Draw a rectangle by dragging the mouse over the area you want to magnify.

Reset Tool

The reset tool restores the selected plot position and zoom level. It does not restore the rotation on 3D plots.

To select the reset tool:

- Click the toolbar button.

Deleting Plots

To delete selected plots:

- Choose **Data > Delete Selected Plots** or press Ctrl+D.

NOTE Deleting a plot does not delete the associated dataset. To delete the datasets or the plots or both, choose **Data > View Info Loaded** (see Figure 2 on page 13).

Performance Options

Working with complex 2D and 3D plots can be sometimes slow. To improve this, Sentaurus Visual provides two options to work faster with plots.

Fast Draw (3D Plots Only)

This option draws only the boundaries of a 3D plot when it is manipulated, which has a large impact on performance.

To enable fast draw:

- Choose **View** > **Fast Draw** or click the toolbar button.

Subsampling (2D and 3D Plots Only)

This option reduces the data points when manipulating a plot, enabling better performance on slower computers.

To enable subsampling:

- Choose **View** > **Subsampling**.

Advanced Options (XY Plots Only)

You can customize the minimum value to be displayed in log scale for xy plots. This configuration can be changed in the User Preferences dialog box (**Edit** > **Preferences**) in the Curve pane (see Figure 13 on page 32).

You can change the minimum value using the **Min Plot Value** field. By default, this value is 1e-20. The minimum value is 1e-300. If you specify a value less than the minimum, the value will be rounded up to the minimum value.

The Performance group box is available to improve the performance of Sentaurus Visual when displaying large `.plt` files. You can reduce the number of points used to define a curve displayed in a plot, which decreases the time taken to draw a curve with a large number of points. You must define two fields:

- The **Points/Curve** field specifies the lower limit at which to activate the Level of Detail algorithm when displaying a curve. The default value is 5000. This means that any curve containing more than 5000 points will switch on this algorithm when displaying the curve. Any curve with fewer than 5000 points will not use this algorithm.
- The **Pixels/Point** field defines the distance in pixels where only one point will be displayed. The default value is 5. This means that any point that is *more than 5 pixels away* from an already plotted point will be plotted. Any point that is less than 5 pixels away from an already plotted point will not be plotted.

NOTE For large `.plt` files, using the Level of Detail algorithm will increase performance, but the accuracy of the drawn curves will decrease.

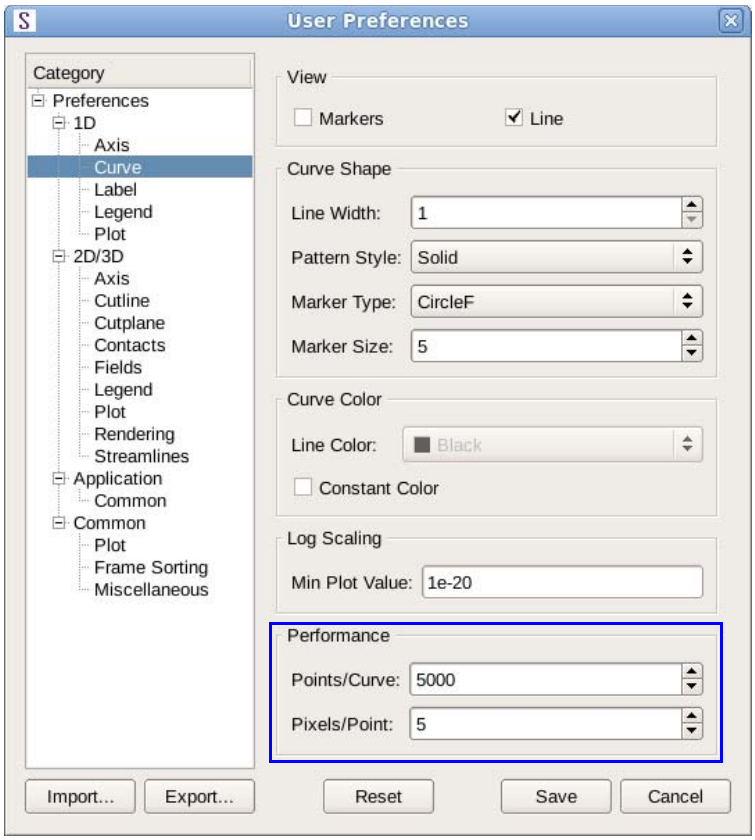


Figure 13 Options for xy plots in User Preferences dialog box

Advanced Options (2D and 3D Plots Only)

Advanced configuration of rendering can be changed in the user preferences (choose **Edit** > **Preferences**). In the User Preferences dialog box, in the Category pane, click **Rendering**. Figure 14 on page 33 shows the rendering fields available.

Advanced options include setting the rendering delay after a mouse operation, modifying the quality of the subsampled interactive structure, and enabling the caching functionality for fields:

- The **End Mouse Interaction Render Delay** field adjusts the delay after interacting with the structure in subsampling or fast draw mode, to redraw the detailed geometry.
- The **Subsampling** option enables the structure to be rendered with fewer points, which optimize the interactive performance with little degradation of the rendering quality.
- If you select the **Automatic** option, Sentaurus Visual automatically renders the subsampled structure. When you select the **Automatic** option, the value in the **Factor** field is used to

fine-tune the algorithm to either performance or quality. A higher value means a higher quality subsampled structure, and a lower value means a lesser quality structure but with better interactive performance.

- If you do not select the **Automatic** option, you can use the Performance/Quality slider to manually choose the quality of the subsampled structure. Moving the slider to the left prioritizes interactive performance, or moving the slider to the right prioritizes rendering quality.
- The **Enable Fast Draw** option enables drawing of the boundaries only of 3D structures.
- The **Enable Field Caching** option helps you to obtain faster transitions between different field visualizations. When it is selected, this option avoids the recalculation of visualization field data that has already been loaded and for which its configuration has not changed.
- The **Disable Drawing** option helps you to improve the loading of files in Sentaurus Visual. This option switches off plot drawing when loading several files and then switches it on when the loading of files is finished.

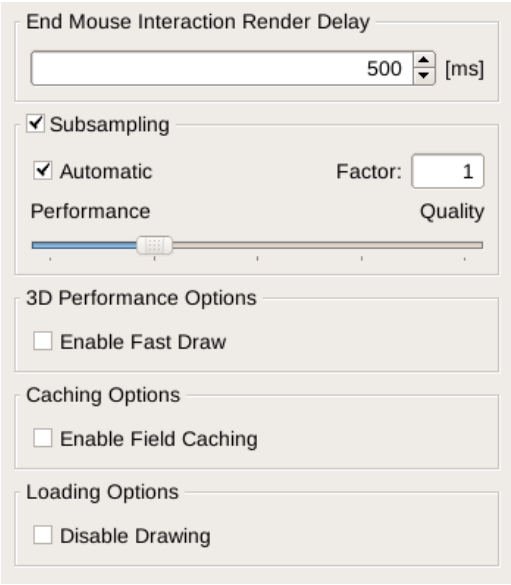


Figure 14 Performance options

The effects of rendering the structure with subsampling or with the **Enable Fast Draw** option selected are shown in Figure 15 on page 34.

NOTE These changes are active only when in GUI mode. When the operation is completed, the full rendering is shown.

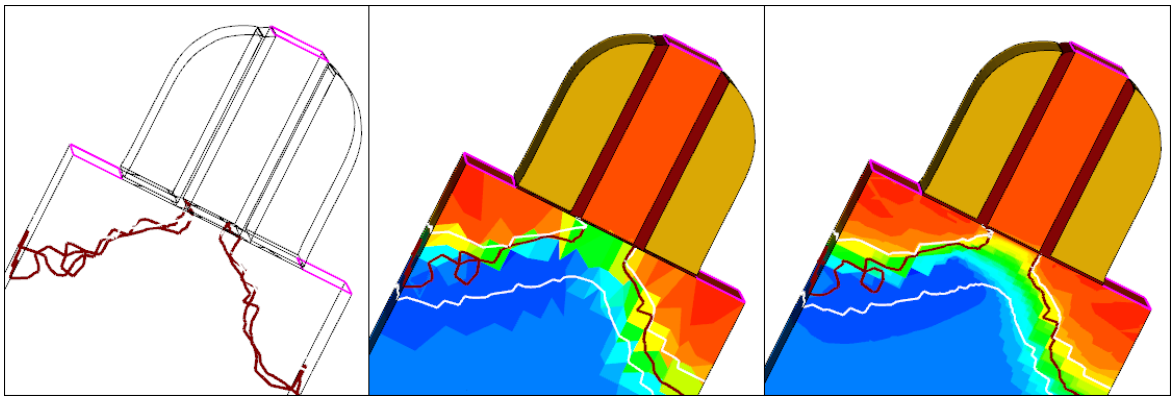


Figure 15 (*Left*) Fast draw enabled, (*middle*) subsampling selected, and (*right*) original structure

Selecting Log Files

By default, Sentaurus Visual generates the standard Tcl log file (`SVisualTcl.log`) with all the commands executed during a session. This log file does not store Tcl commands executed by a script or procedure. If this log file already exists, Sentaurus Visual creates a backup file called `SVisual.log.BAK`.

In addition, by default, another Tcl log file is created if Sentaurus Visual is executed from the command line (or Sentaurus Workbench) with a script, for example:

```
% svisual scriptFile.tcl
```

This additional log file not only stores Tcl commands executed during the session, but also writes Sentaurus Visual Tcl commands executed from a script. In this case, the log file contains more detailed information than the standard Tcl log file.

This additional log file is named according to the script executed from the command line (or Sentaurus Workbench), changing the file extension of the script from `.tcl` to `.log`, for example, `scriptFile.log`.

To change the selection of log files:

1. Choose **Edit** > **Preferences**.
2. In the User Preferences dialog box, expand **Application** > **Common** (see Figure 16 on page 35).

3. Under Tcl Logging, change the selected options as required.
4. Click **Save**.

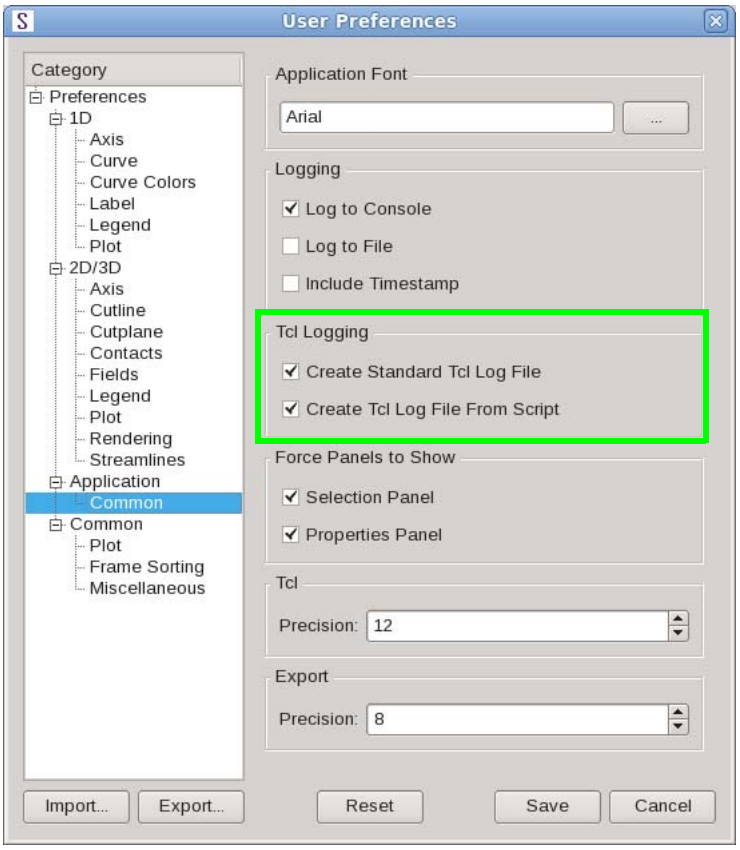


Figure 16 User Preferences dialog box showing options for Tcl log files

CHAPTER 4 Working With XY Plots

This chapter presents specific topics about working with xy plots in Sentaurus Visual.

Loading XY Plots

Loading an xy file does not automatically plot the dataset associated with it. Instead, the loaded datasets appear in the Selection panel, and a blank plot is created as shown in Figure 17.

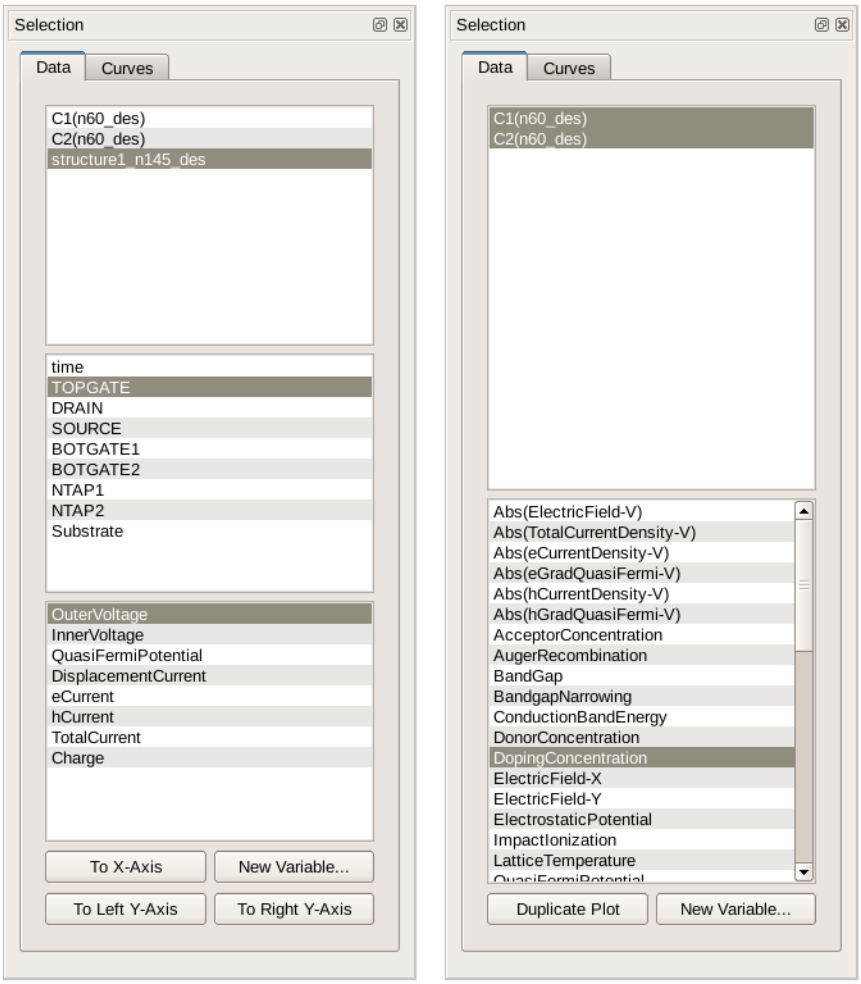


Figure 17 Selection panel showing (*left*) active datasets of xy plot and (*right*) active datasets of cutline plot

The top pane corresponds to the datasets loaded, the middle pane shows the variables present in the selected dataset, and the bottom pane lists the composite variables available in the middle pane.

NOTE For `.plx` files and cutline plots, the x-axes and y-axes are assigned automatically, and the respective curve is generated onto the active plot.

Plotting One Curve

To plot an xy curve, you must select a dataset, and then assign the x-axis and y-axis variables from the available options in the middle pane (or bottom pane if one variable is a composite). The result of selecting *vd* as the x-axis variable and *ib* as the left y-axis variable from the `vd_ib_vb0_vg0.6_vs0` dataset can be seen in Figure 18.

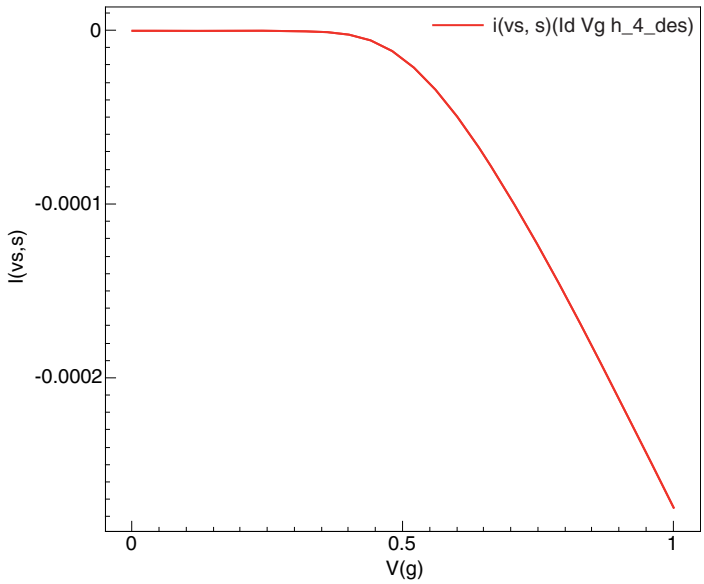


Figure 18 Plotting a single xy curve

Plotting Multiple Curves

To display multiple datasets on the same plot:

1. Hold the Ctrl key and click the required datasets.

   The common variables in the datasets selected are displayed in the middle pane and bottom pane.

2. Repeat the procedure in the same way as for plotting one curve.

As shown in Figure 19, five datasets were selected; *v(g)* was selected as the x-axis variable and *id(vd,d)* was selected as the left y-axis variable.

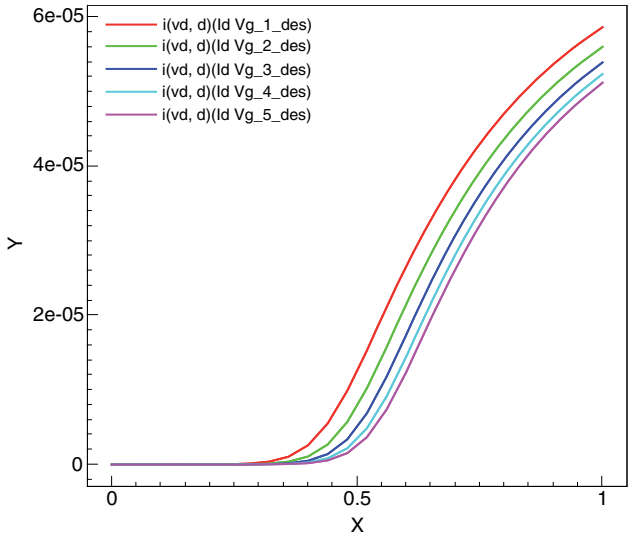


Figure 19 Plotting multiple xy curves

When you display multiple curves in an xy plot, the curves are colored according to user-defined rules set in the User Preferences dialog box (expand **1D > Curve Colors**). By default, Sentaurus Visual displays curves using a round-robin logic from a list of colors as shown in Figure 20 on page 40.

To add a custom color to the list of colors:

1. Under Curve Color Behavior, select **List Colors**.
2. Under Curve Color Selection, click the **Add** button.
3. In the dialog box that opens, specify a custom color.
4. Click **OK** to close the dialog box.
5. Click **Save**.

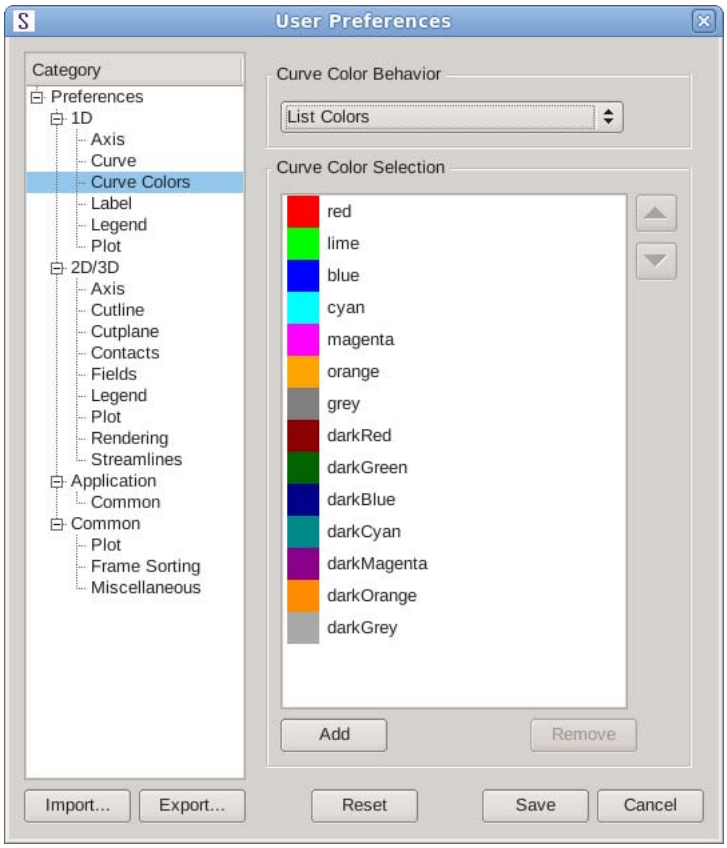


Figure 20 List of default curve colors

To set the same color for all curves in xy plots:

1. Under Curve Color Behavior, select **Constant Color** (see Figure 21 on page 41).
2. Under Curve Color Selection, select a color from the list.
3. Click **Save**.

You can also map colors to curves using curve names as well as wildcards. The color-to-curve mapping rules are applied from top to bottom.

To map colors:

1. Under Curve Color Behavior, select **Map Colors**.
2. Under Curve Color Selection, specify the mapping as required (see Figure 22 on page 42).
3. Click **Save**.

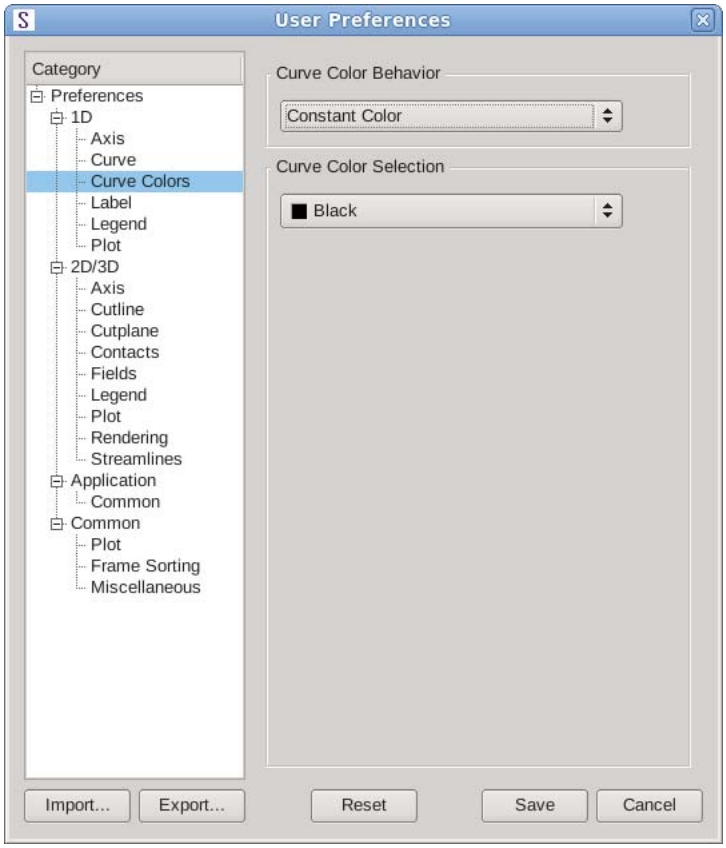


Figure 21 User Preferences dialog box showing a constant curve color has been specified

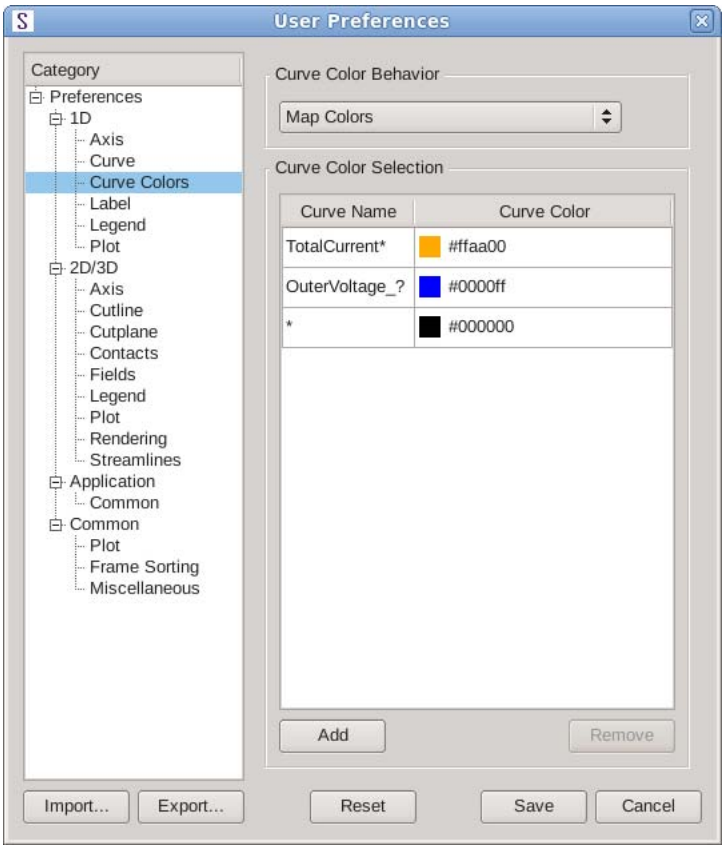


Figure 22 Mapping curve colors

Visualizing Multiple TDR States

TDR files can contain multiple states of different simulation results. The states are related with regard to geometry. In other words, the main structure data (the number of points) is maintained in all states but their variable data changes. Sentaurus Visual allows you to visualize all the states.

For xy plots, variables can have multiple states. If a curve is created using such variables, a navigation area specific to the existing curve is displayed to allow you to easily navigate through the different states of the curve (see Figure 23 on page 43).

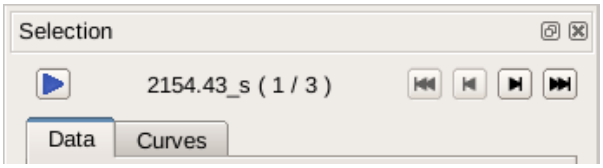


Figure 23 Navigation area displaying state name and state index of curve

The navigation area allows you to switch between displayed states quickly with the:

- Next State button
- Previous State button
- First State button
- Last State button

With any change to the state index, the plot title will be updated reflecting the state name.

In addition, the Play button allows you to automatically go through all the states, with a 1 second delay between changes, in ascending order. If the last state is reached and the Play button is still active, the sequence will restart.

If displayed curves do not have the same states, the navigation area changes to display the generic state name *state* and the state index of all curves (see Figure 24).

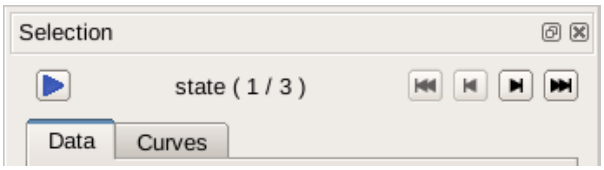


Figure 24 Navigation area displaying state index of curve

When displaying more than one curve with different state lengths, if you increase the state index to be displayed and one of the curves already reaches its maximum state number, that curve will remain in the maximum valid state number and only the other curves will continue changing accordingly. Switching the state of a curve is a property of a plot, but it cannot be handled in isolation for a single curve being plotted with other curves with different state lengths.

Plotting a multistate curve together with non-multistate (normal) curves will still display the navigation area and the navigation of the multistate curves as previously described. However, normal curves are not affected by any state changes.

Cutline Plots

Cutline plots have a special interface that allows you to plot new curves by simply selecting one or more variables from a single dataset or a set of datasets. The curve visualization depends only on the datasets and variables selected in the Selection panel (see Figure 25 and Figure 26 on page 45). You only need to select a new variable (or a set of them) to remove the old curves and to create new ones.

The Selection panel does not have the buttons to assign variables to the x-axis or y-axis, but it maintains the **New Variable** button and implements the **Duplicate Plot** button that is used to duplicate the current plot as an xy plot, which enables the features of an xy plot for the currently displayed cutline plot by cloning it.

Cutline plots have a special plot title that follows the format: Cutline_* Plot, where * can be X, Y, Z, or Free, depending on the type of cut. This helps to distinguish cutline plots from xy plots.

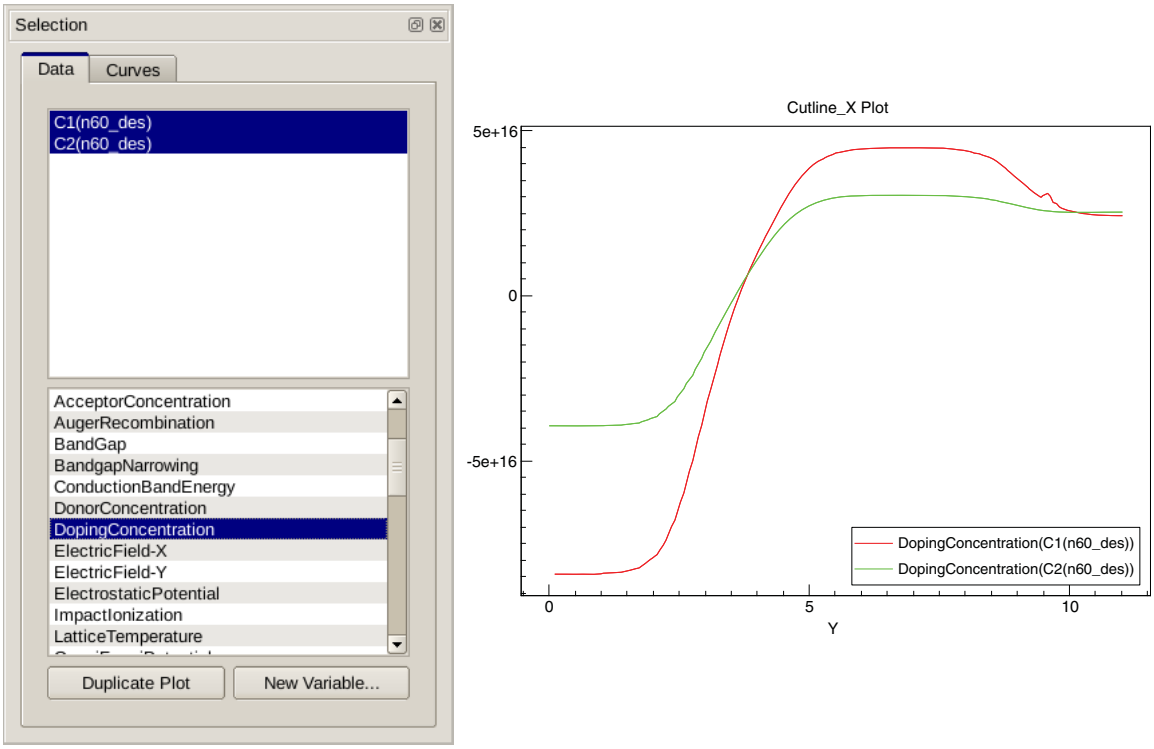


Figure 25 Cutline plot displaying DopingConcentration from two datasets

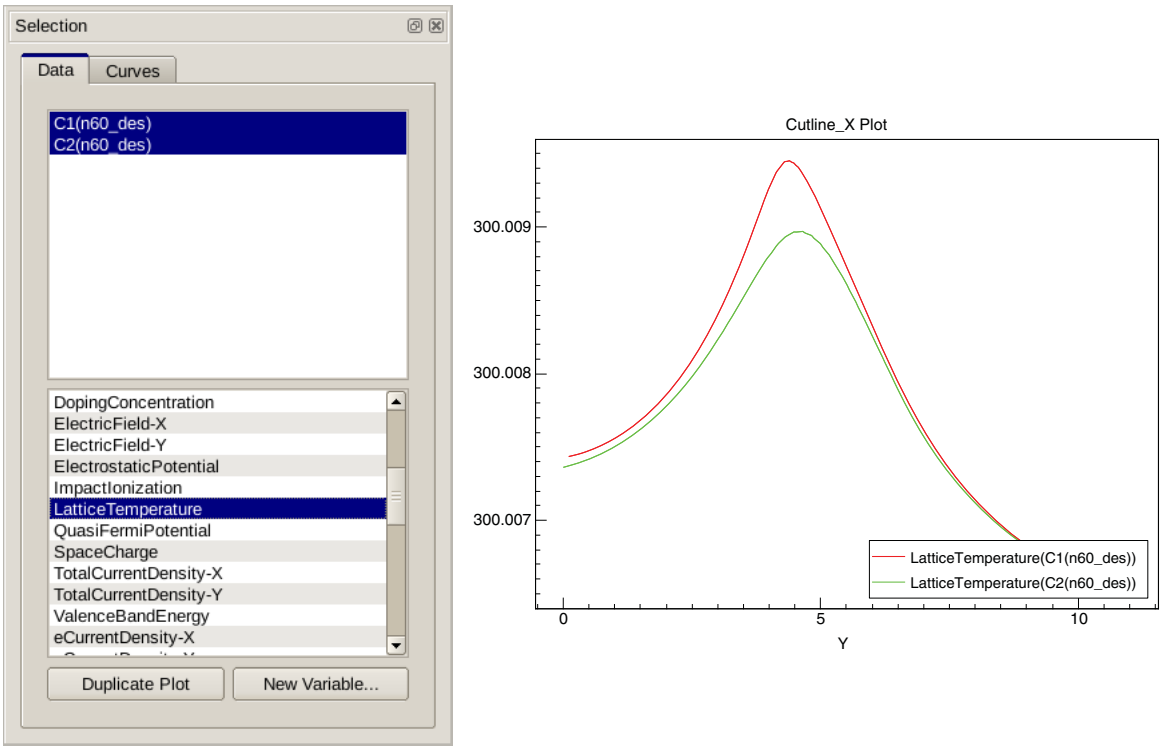


Figure 26 Cutline plot displaying LatticeTemperature from two datasets

Curve Properties

To edit the properties of a curve, select it from the active plot, or you can select the curve from the list in the Selection panel. You also can select multiple curves in the Selection panel and apply properties to all of them. The Curve Properties panel is displayed (see Figure 27 on page 46).

In the Curve Properties panel:

- On the **Main** tab, you can change the label of the curve, and select to show or hide the legend and named curve.
- On the **Shape** tab, you can change properties such as curve color, line style, line width, and data pointers.

- On the **Trans.** tab, you can apply curve transformations. It is possible to apply an integration or the first and second derivative to the dataset, or to plot a function using the dataset values to evaluate the required function. In addition, you can shift and scale the selected curve in the x-axis and y-axis.
- On the **Analysis** tab, you can perform certain analyses on the dataset. For a detailed explanation, see Analysis Tool on page 56.

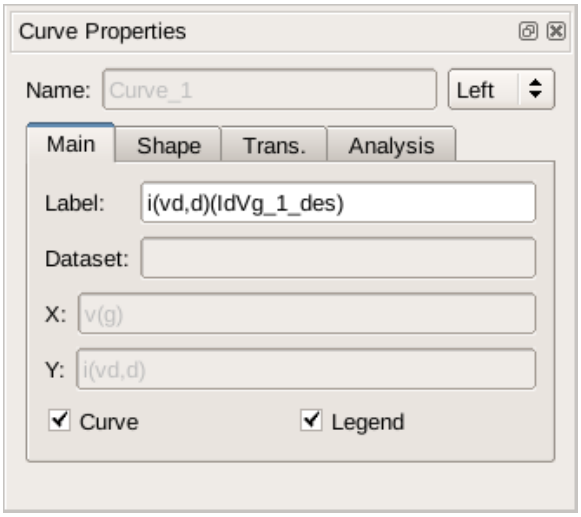


Figure 27 Curve Properties panel

Modifying Properties in Multiple Curves

Sentaurus Visual provides a dialog box where you can modify all curve properties in one view (see Figure 28 on page 47). You can modify one property in several curves at the same time.

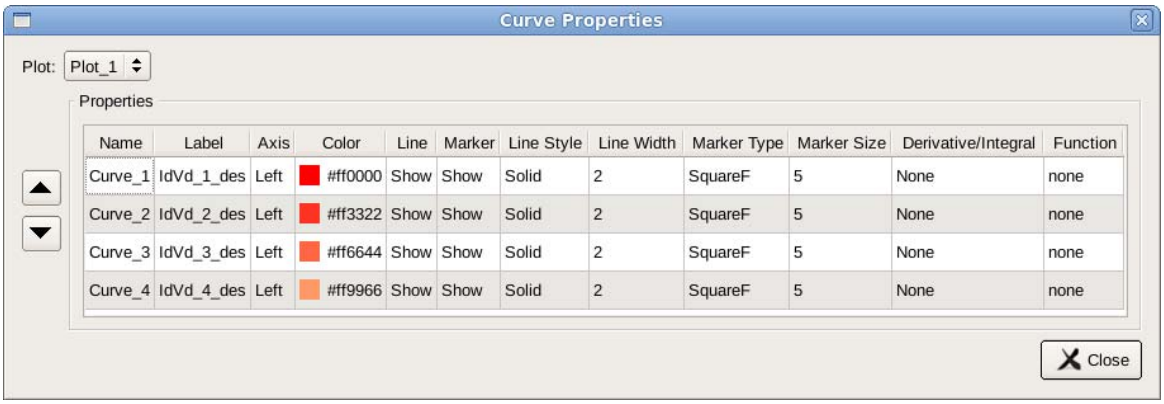
To modify a property in multiple curves:

1. Choose **Data** > **Curve Properties**, or click the toolbar button.
2. Select the required curve rows.
3. Click the column header of the property you want to modify.

   A dialog box is displayed where you change the value of the property.

In addition, you can change the order of curves. To do this, select one or more curve rows, and click either the Up arrow button or the Down arrow button at the left of the dialog box.

The order of curves changes immediately and is displayed in the legend as well as the list of curves in the Selection panel.



Name	Label	Axis	Color	Line	Marker	Line Style	Line Width	Marker Type	Marker Size	Derivative/Integral	Function
Curve_1	IdVd_1_des	Left	#ff0000	Show	Show	Solid	2	SquareF	5	None	none
Curve_2	IdVd_2_des	Left	#ff3322	Show	Show	Solid	2	SquareF	5	None	none
Curve_3	IdVd_3_des	Left	#ff6644	Show	Show	Solid	2	SquareF	5	None	none
Curve_4	IdVd_4_des	Left	#ff9966	Show	Show	Solid	2	SquareF	5	None	none

Figure 28 Curve Properties dialog box

Plot Area Properties

The appearance of the plot area can be modified using the Plot Properties panel (see Figure 29). The Plot Properties panel allows you to change such attributes as the background and foreground colors of the plot, and to show or hide the title, legend, axes, or curves.

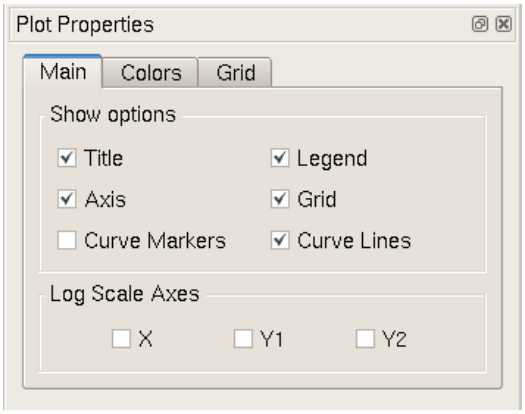


Figure 29 Plot Properties panel showing selected options

For example, to hide the legend:

1. Select the plot.
2. On the **Main** tab, clear the **Legend** option.

NOTE To show the Plot Properties panel, double-click an empty part of the required plot if another panel is active.

See Quick Access to Tabs of Plot Properties and Axis Properties Panels on page 8.

Legend Properties

Legend properties such as position, font attributes, and colors can be changed in the Legend Properties panel (see Figure 30). To open the panel, double-click the legend of an xy plot.

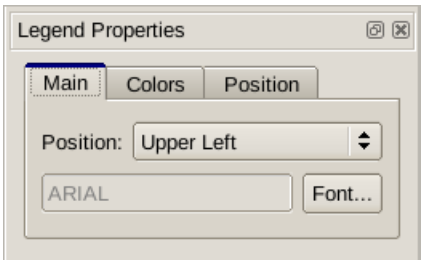


Figure 30 Legend Properties panel

Axis Properties

The appearance of axes can be modified using the Axis Properties panel (see Figure 31).

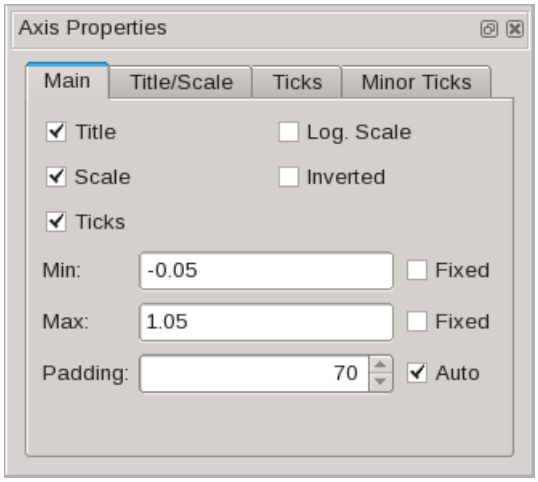


Figure 31 Axis Properties panel showing selected options

To open the Axis Properties panel, double-click any axis in the plot area.

See Quick Access to Tabs of Plot Properties and Axis Properties Panels on page 8.

Changing the Axis Padding

You can change the padding value using the **Padding** field on the **Main** tab of the Axis Properties panel. By default, the **Auto** option is selected for padding, in which case, the padding value is calculated automatically and cannot be edited.

When several xy plots are linked and the **Auto** option is selected, the padding for each axis is the same for all linked plots. The padding value used is the *largest* padding value of each axis from all linked plots. This feature helps to compare curves or plots visually.

Changing the Axis Precision

You can set the precision of the axis (for xy plots and 2D plots) on the **Title/Scale** tab of the Axis Properties panel (see Figure 32). The precision refers to the number of *relevant* digits after the decimal point.

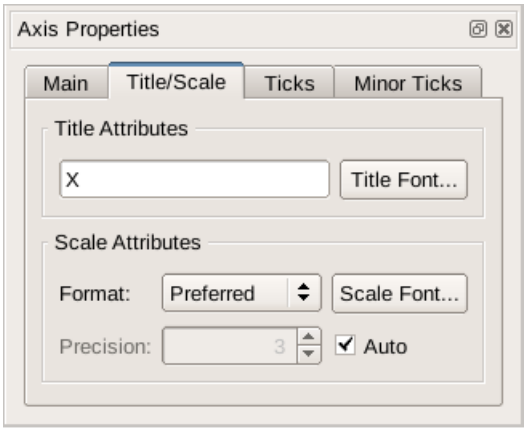


Figure 32 Axis Properties panel showing the Title/Scale tab where you can change the precision of the axis manually

By default, the precision is chosen automatically based on the dimension of the plot, but this can be manually changed:

1. Clear the **Auto** option.
2. In the **Precision** box, select the precision required.

Duplicating XY Plots

You can duplicate an xy plot by choosing **Data > Duplicate Plot**. All properties of the selected plot are replicated in a new plot.

Using Symbols and Scientific Notation in Plots

You can insert Greek symbols, subscripts, superscripts, and math symbols in xy plots by using XML tags in the text box of the plot title, the axis labels, and the legend. The available tags are:

Symbol	Tag	Example	Result
Greek symbol	`<greek>`	`<greek>abcdefgh</greek>`	αβχδεφγη
Math symbol	`<math>`	`<math>plusminus</math>`	± See Table 3.
Subscript	`<sub>`	`V<sub>d</sub>`	V_d
Superscript	`<sup>`	`10<sup>-8</sup>`	10^{-8}
Bold	`<b>`	`<b>word</b>`	**word**
Italic	`<i>`	`<i>word</i>`	*word*
Underline	`<u>`	`<u>word</u>`	word
Strikethrough	`<s>`	`<s>word</s>`	~~word~~

Table 3 lists the defined words that are allowed in the `<math>` tag. Only one word is allowed in the `<math>` tag.

Table 3 Defined words that are allowed in <math> tag

Word	Result	Word	Result
`3root`	$\sqrt[3]{\ }$	`laplace`	ℒ
`4root`	$\sqrt[4]{\ }$	`mho`	℧
`contains`	∋	`notcontains`	∌
`contourintegral`	∮	`notelementof`	∉
`deriv`	∂	`notexists`	∄
`doubleintegral`	∬	`permille`	‰
`e`	*e*	`permyriad`	‱
`elementof`	∈	`plusminus`	±

Table 3 Defined words that are allowed in <math> tag

Word	Result	Word	Result
emptyset	$\varnothing$	sqroot	$\sqrt{}$
exists	$\exists$	sum	$\sum$
forall	$\forall$	surfaceintegral	$\oiint$
fourier	$\mathcal{F}$	tripleintegral	$\iiint$
gradient	∇	union	$\cup$
inf	∞	volumeintegral	$\oiiint$
integral	$\int$		

Figure 33 shows an example of how these symbols are displayed.

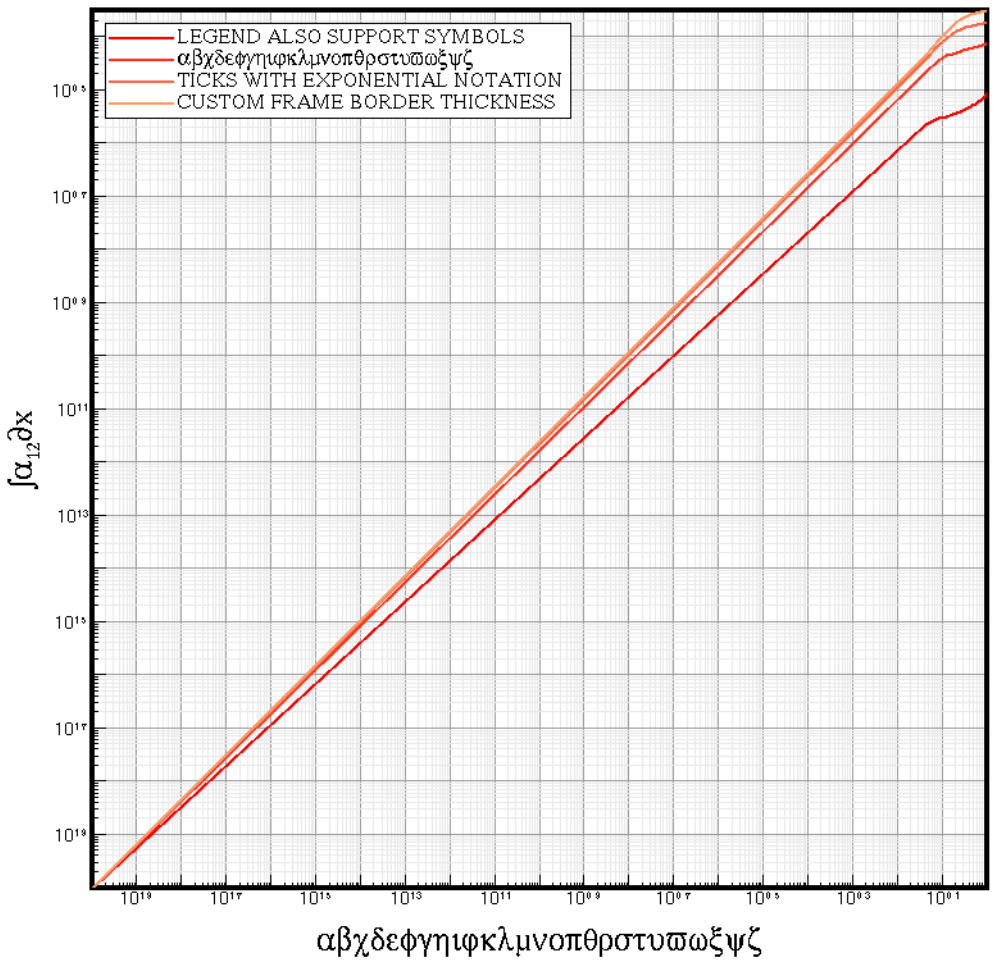


Figure 33 Plot showing Greek symbols, math symbols, and scientific notation in axis labels and legend

You also can use scientific notation in the axis labels of xy plots by choosing **Scientific** from the **Format** list on the **Title/Scale** tab of the Axis Properties dialog box (see Figure 34).

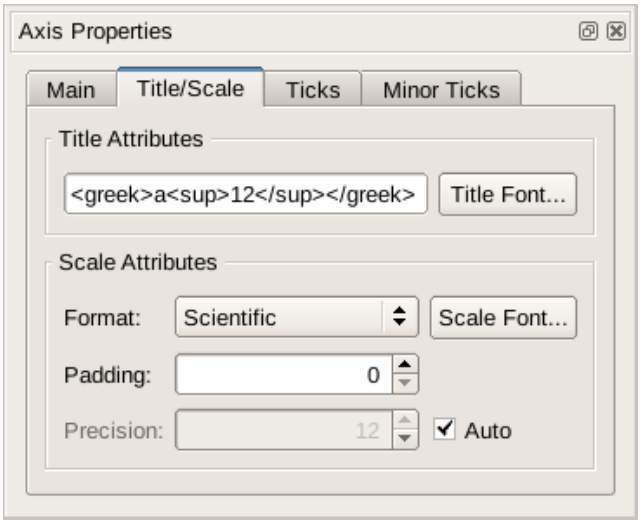


Figure 34 Axis Properties dialog box showing the selection of scientific notation

Best Look Option

Best look is a useful option to select automatically the optimal parameters for the active plot. This option:

- Adjusts the zoom level to the optimal position and changes the x-axis label to the variable used if it is common between curves.
- Changes the label of the legend to the variable being plotted (if there is only one curve, it disables the legend and uses the variable name for the y-axis label).
- Changes the title to the dataset name if all curves share the same dataset.

To enable best look:

- Click the  toolbar button.

Plotting Band Diagrams

Sentaurus Visual allows you to plot band diagrams, which show the electron energy of the valence band and the conduction band edges versus a spatial dimension. Figure 35 shows an example of a band diagram.

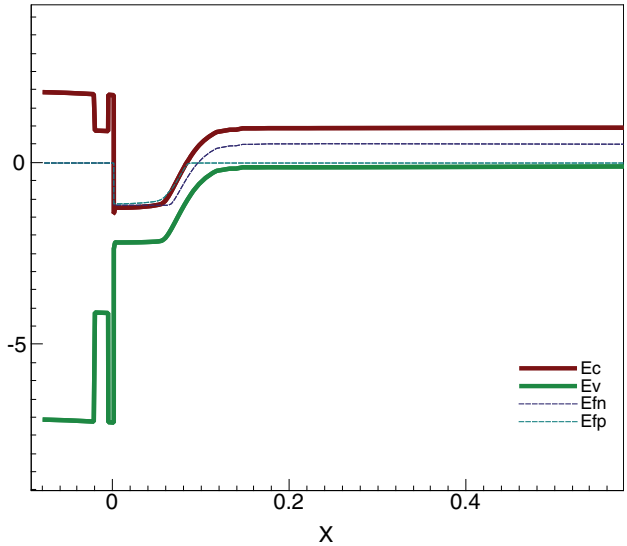


Figure 35 Example of a band diagram

To create a band diagram for datasets, click the  toolbar button.

The dataset must have the following variables defined (variable names are italicized):

- The conduction band energy (*ConductionBandEnergy*)
- The valence band energy (*ValenceBandEnergy*)
- The electron quasi-Fermi energy (*eQuasiFermiEnergy*) or the electron quasi-Fermi potential (*eQuasiFermiPotential*) but not both in the same dataset
- The hole quasi-Fermi energy (*hQuasiFermiEnergy*) or the hole quasi-Fermi potential (*hQuasiFermiPotential*) but not both in the same dataset

NOTE Typically, band diagrams are created from xy datasets resulting from cuts of 2D or 3D geometries.

Saving the Plot to a Tcl File

Sentaurus Visual can save the current xy plot (including the plot settings, curve data, and displayed curves) to a Tcl file that allows you to recreate the plot easily. You can either choose **Data** > **Save Plot** or use the `save_plot_to_script` command (see save_plot_to_script on page 257).

When the file is generated, you can either choose **File** > **Run Tcl Script** or use the `load_script_file` command (see load_script_file on page 239) to reproduce a previously saved plot.

The generated file has a particular structure that you can edit for customized loads:

- *Plot Configuration Module*: This module updates the plot with the saved properties, which is performed with the `set_axis_prop`, `set_grid_prop`, `set_legend_prop`, and `set_plot_prop` commands (the `set_axis_prop` command is executed for x-axes, y-axes, and y2-axes).
- *Curves Configuration Module*: This module updates the curves with their saved properties such as color and line width. The update is performed with the `set_curve_prop` command.
- *Drawings Restoration Module*: This module places the drawings in the plot and restores their properties.

  NOTE Only text box properties are restored. Other drawings properties must be updated manually.
- *Data Initialization Module*: This module initializes the curve data using a Tcl array structure. The final data is stored as:

  ```
  set datasetList(<curveName>) { {<xAxisData>} {<yAxisData>} }
  ```
- *Plot and Curves Restoration Module*: After its initialization, the data module creates the xy plot and then creates the curves using the `datasetList` data. In general, this module should never be modified.

Probe Tool

The probe tool allows you to sample the intersection value for a horizontal or vertical line depending on whether the probing is performed on the x-axis or y-axis. In xy plots, the probe tool uses the interpolation that matches the axis to obtain the value: linear when the axis is in normal mode and log when the axis is in log mode.

To use the probe tool, click the  toolbar button.

Probe Options

In the Probe panel, the following options are available:

- To show the active curve only, select the **Only Active Curve** option.
- To show guide lines while probing, select the **Show Guide Lines** option.

Calculate Scalar Tool

The calculate scalar tool allows you to perform complex mathematical operations on the available 1D data in memory. To display the Calculate Scalar dialog box, choose **Tools** > **Calculate Scalar** (see Figure 36).

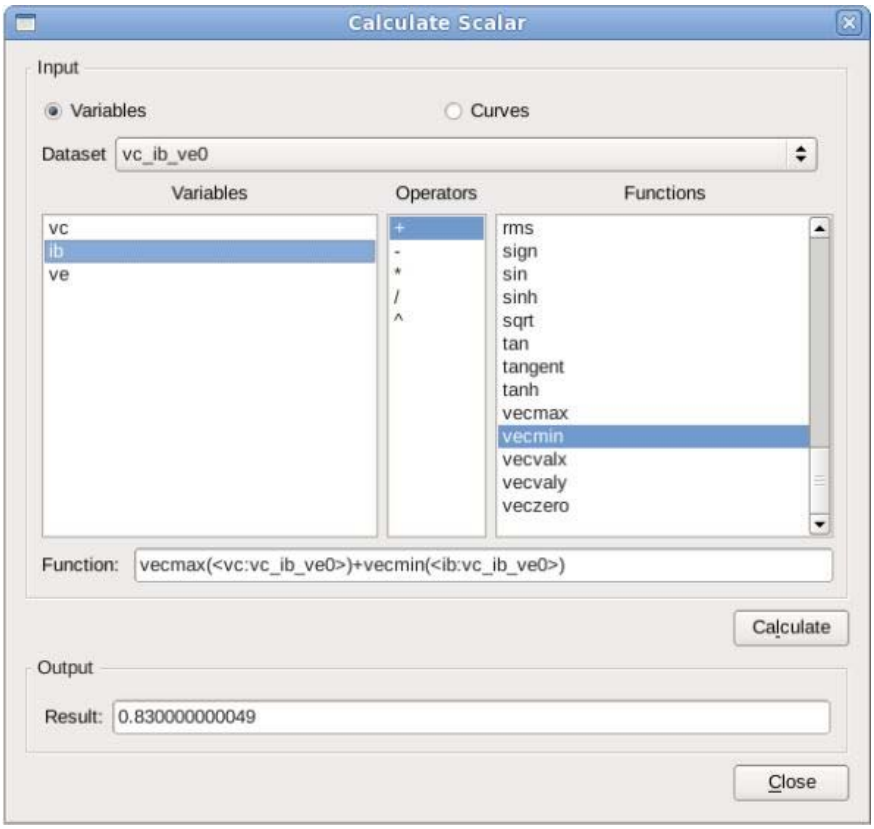


Figure 36 Calculate Scalar dialog box showing the results from the mathematical operations in the Function field

In the dialog box, you create a formula by inserting functions and operators, and using existing 1D data. For the latter, you must select whether the formula will operate on variables (from a dataset) or curves (from a plot).

NOTE The last operation that encloses the entire set of functions must be a scalar value function. Otherwise, the calculation will fail.

Analysis Tool

The analysis tool allows you to compute the electrical characteristics of field-effect transistors. Depending on the curve being plotted, different analyses can be performed, such as the threshold voltage, the maximum transconductance value, the drain saturation current, the leakage current, and the output resistances in the linear or saturation region.

To enable the analysis tool, click the  toolbar button.

Table 4 lists the available curve analyses.

Table 4 Types of analysis

Type of analysis	Description
V_{th}	Threshold voltage is defined as the minimum gate electrode bias required to strongly invert the surface under the poly and to form a conducting channel between the source and the drain regions. It can be calculated on I_d–V_g curves.
$G_{M(MAX)}$	Transconductance is a measure of the sensitivity of the drain current to changes in the gate–source bias. It is influenced by gate width, which increases in proportion to the active area as cell density increases. It can be calculated on I_d–V_g curves.
$I_{D(SAT)}$	For a constant gate voltage (V_g), this computes the drain saturation current on I_d–V_d curves.
$I_{D(OFF)}$	For a constant drain voltage (V_d) and a gate voltage (V_g) equal to zero, this computes the leakage drain current on I_d–V_g curves.
R_{out}	R_{out} is the value of the output resistance in the saturation region when $V_g > V_{th}$. This value can be calculated on I_d–V_d curves.
R_{on}	R_{on} is the value of the on-state resistance. It is calculated when the transistor is in the linear region. This value can be calculated on I_d–V_d curves.

For more information about the extraction formulas used to obtain the results in the analysis tool, see Inspect User Guide, Chapter 8 on page 69.

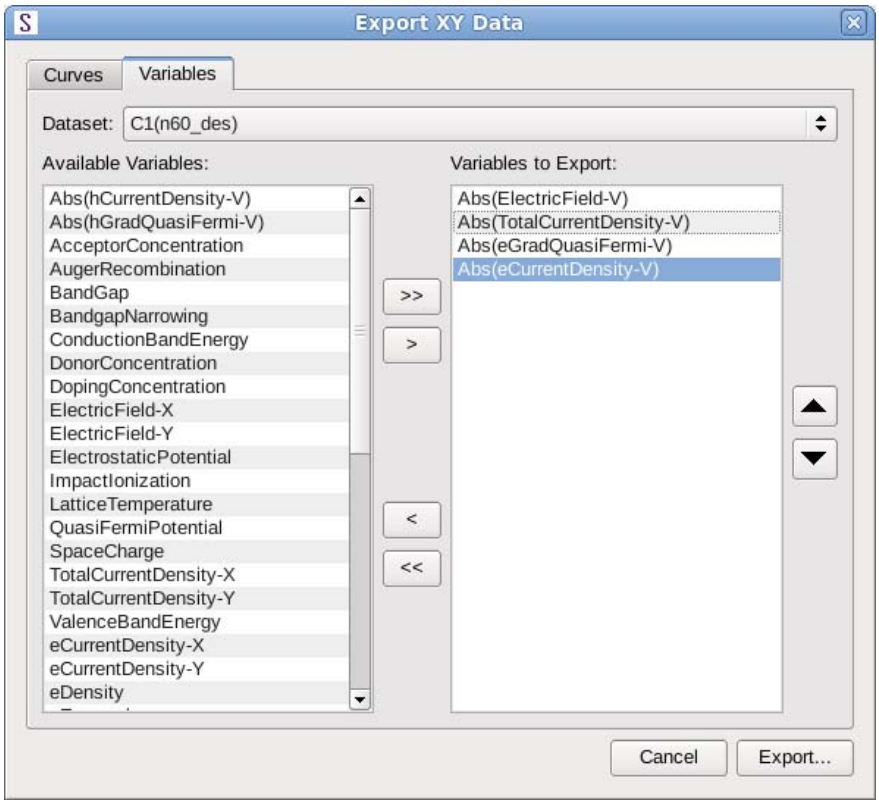
Exporting Data From Variables and Curves

You can export data from variables to a `.csv` file and data from curves to a `.csv` or `.plx` file, to allow the use of other tools for further analysis and plotting.

To export data from a variable or curve:

1. Select an xy plot.
2. Choose **Data** > **Export XY Data**, or click the toolbar button.

   The Export XY Data dialog box is displayed.



3. Select the variables or curves to export by clicking the relevant tab.
4. Export all variables (click the **>>** button), or export only the variables you need (click the **>** button) to the Variables to Export pane.
5. Define the order of variables or curves in the export list using the **Move Up** or **Move Down** buttons to the right of the Variables to Export pane.

6. Click **Export**.
7. In the dialog box that is displayed, select the file format in which to export the data.

NOTE The precision of the data exported can be changed in the User Preferences dialog box (expand **Application** > **Common** and, under Export, specify the precision).

NOTE When you export 1D plot variables to a `.csv` file, Sentaurus Visual might add a row to the file. The data in this row is only used internally. This additional row is always the second row and contains only `none` and `SELECTED` values. You can delete this row.

CHAPTER 5 Working With 2D and 3D Plots

This chapter presents specific topics about working with 2D and 3D plots in Sentaurus Visual.

Visualizing 2D and 3D Plots

Sentaurus Visual can visualize simulation results for 2D and 3D plots. When a 2D or 3D file is loaded, Sentaurus Visual automatically generates a plot with the edge, field, and bulk layers activated by default as shown in Figure 37.

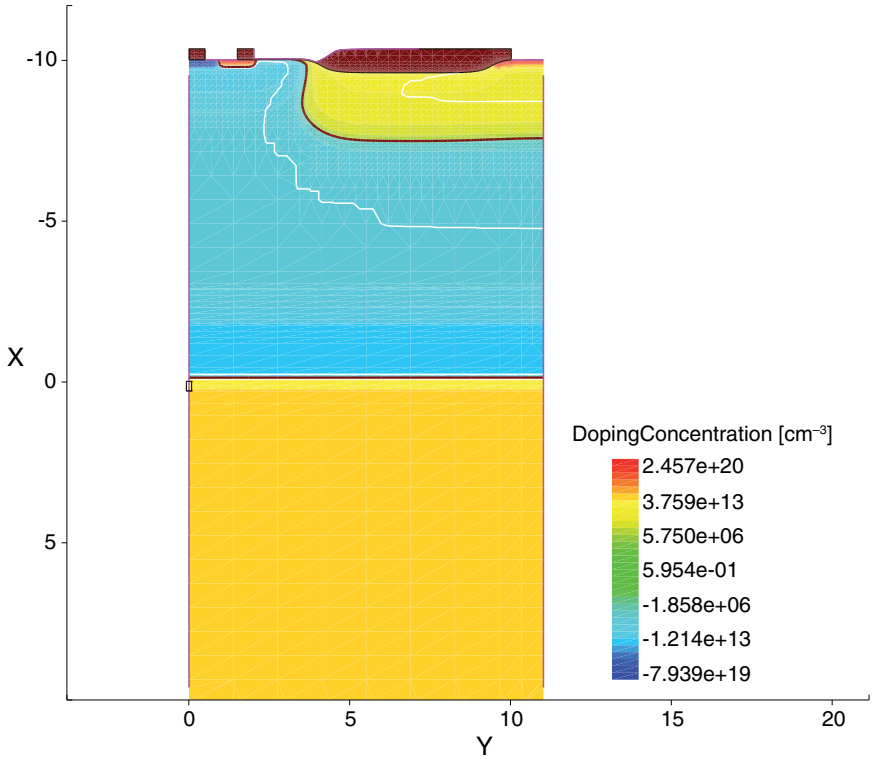


Figure 37 Example of 2D plot

Visualizing Fields

The active field to be visualized in a plot can be chosen in the Selection panel (see Figure 38). The fields can be either scalar or vector. For scalar fields, you can choose the number of colors in which the visualization will be divided as well as the scale, which can be linear, logarithmic, hyperbolic arcsine (Asinh), logarithmic of the absolute (LogAbs), or some custom list of points that you define.

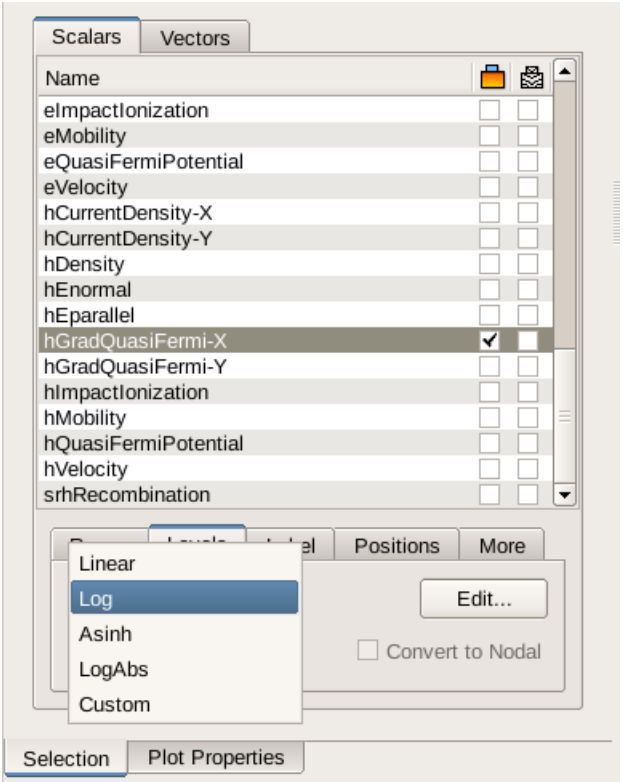


Figure 38 Selection panel showing options for visualizing fields

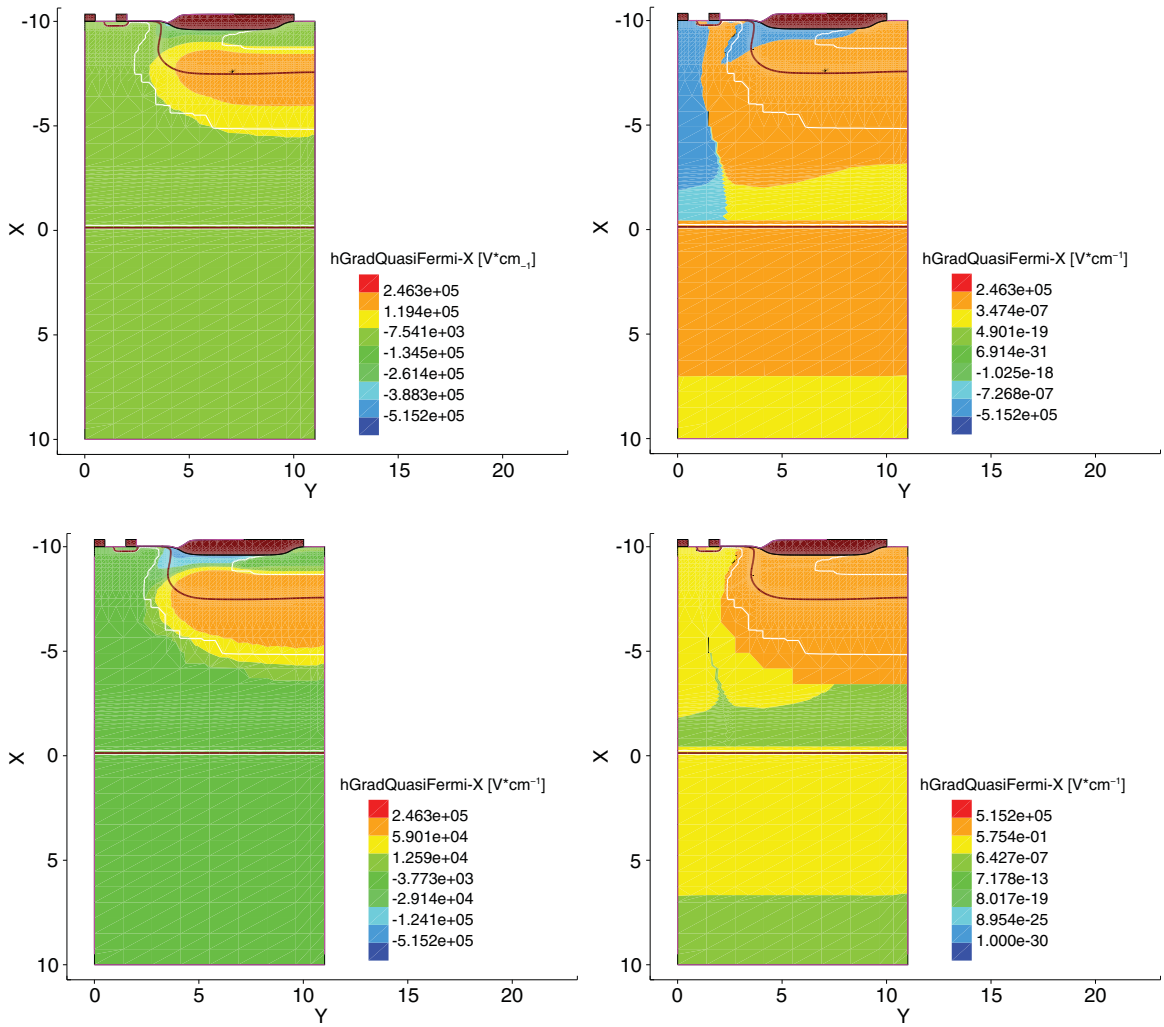


Figure 39 Scale options for field visualization: (*upper left*) linear scale, (*upper right*) logarithmic scale, (*lower left*) Asinh, and (*lower right*) LogAbs

Visualizing Fields Defined on Interface Regions

Structures can have fields defined on interface regions. These fields are distinguished from regular region fields by the prefix Int(*field name*). For example, the name of the field `DopingConcentration` would change to `Int(DopingConcentration)`. The prefix allows you to easily identify such fields on the **Scalars** tab of the Selection panel.

For 2D plots, the width of the interface increases automatically to improve visualization of the field data (see Figure 40 on page 62 (*right*)).

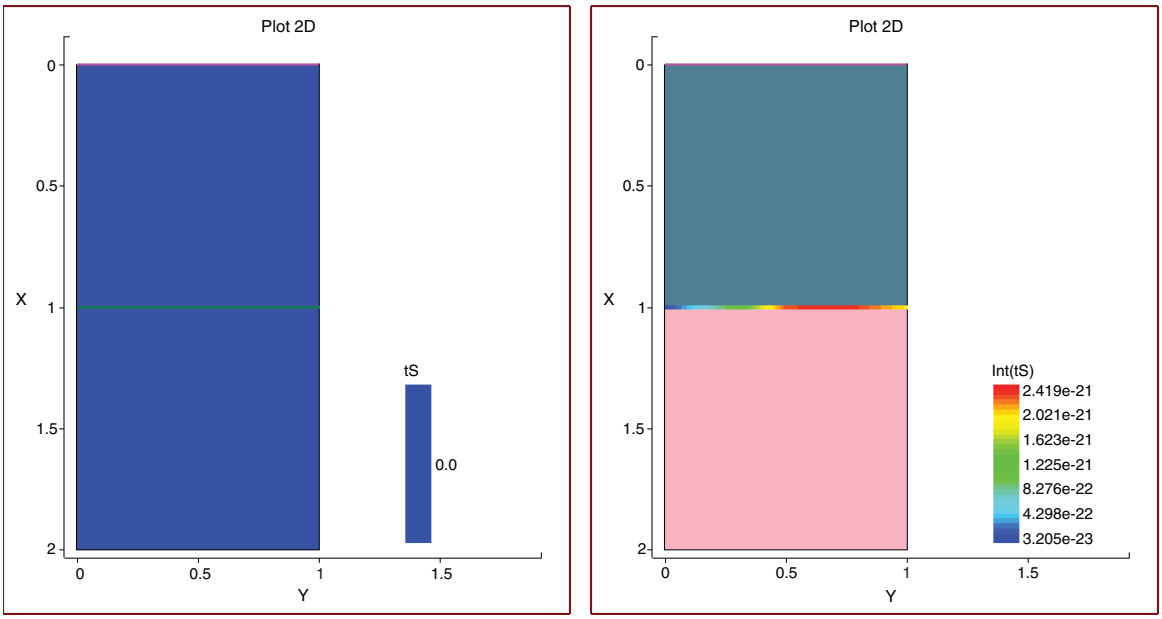


Figure 40 Interface data displayed in 2D plots for: (*left*) regular region field and (*right*) field defined on interface region

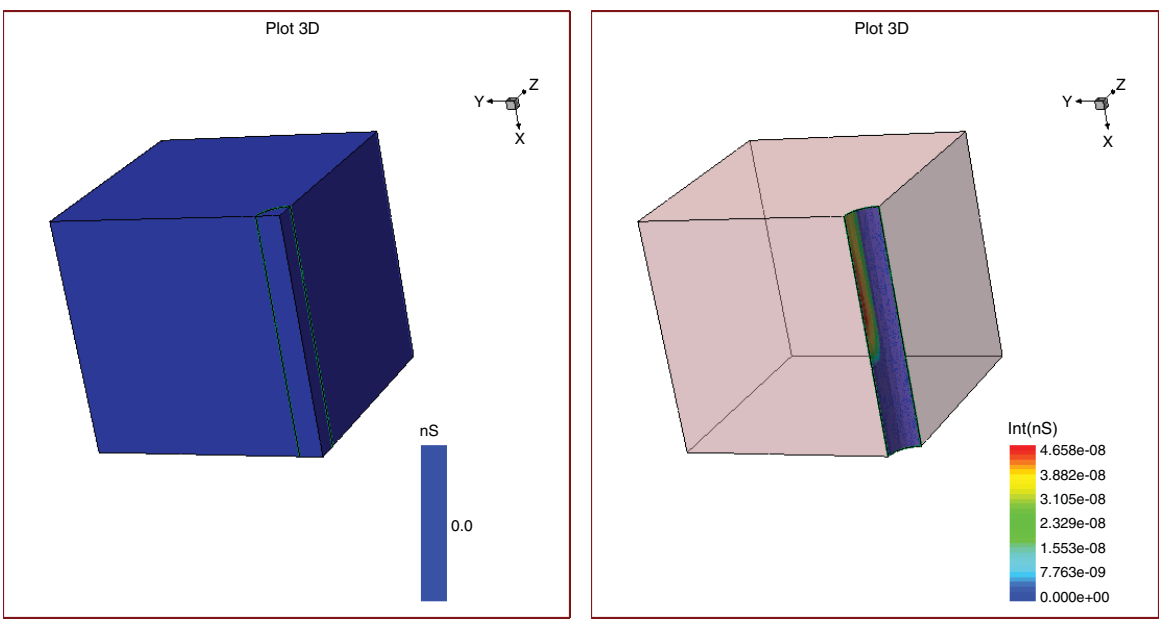


Figure 41 Interface data displayed in 3D plots for: (*left*) regular region field and (*right*) field defined on interface region

NOTE When rendering 3D plots, you can observe the *stitching* phenomenon. It occurs when interface regions share their points with other regions. Both these types of region consist of coplanar polygons where two faces occupy essentially the same space, but neither is in front of the other. The result is a visible *flickering* as affected pixels are rendered from one polygon and then another polygon randomly.

For this reason, when working with 3D plots, you should switch on translucency of other regions to minimize the flickering effect.

Visualizing Automatically Generated Regions

You can visual junction lines and depletion regions.

Junction Lines

Sentaurus Visual calculates automatically the junction line in semiconductor regions where the `Doping` field is present.

The junction line is visualized as a dark-red contour line and is defined where `Doping` (`DopingConcentration` or `NetActive`) is equal to zero (`Doping` = 0).

Depletion Regions

Sentaurus Visual calculates automatically the depletion region in semiconductor regions where the `Doping` field (`DopingConcentration` or `NetActive`) and the electron and hole density fields (`eDensity` and `hDensity`, respectively) are present.

The edge of the depletion region is visualized as a white contour line. The depletion region is defined by:

$$n \cdot \frac{\text{eDensity}}{\text{Doping}} - p \cdot \frac{\text{hDensity}}{\text{Doping}} = \text{DepletionEdgeValue}$$

where:

$$n = \max\left(\frac{\text{Doping}}{\text{abs(Doping)} + 1}, 0\right)$$

$$p = \max\left(\frac{-\text{Doping}}{\text{abs(Doping)} + 1}, 0\right)$$

The `DepletionEdgeValue` is equal to 0.05 by default. You can modify this value by changing the Sentaurus Visual configuration file (`~/.config/Synopsys/SVisual.conf`).

The parameter that defines the `DepletionEdgeValue` is called `depletion\edgeValue` and belongs to the `PlotHD` group:

```
...
[PlotHD]
...
depletion\edgeValue=0.05
...
```

Visualizing Multiple TDR States

TDR files can contain multiple states of different simulation results from Sentaurus Process Kinetic Monte Carlo simulations or Sentaurus Device simulations. The states are related with regard to geometry. In other words, the main structure data and regions are maintained in all states but their field data changes. Sentaurus Visual allows you to visualize all the states.

For 2D and 3D plots that are generated from TDR files with multiple states, a navigation area specific to such plots is displayed to allow you to easily navigate through them (see Figure 42).

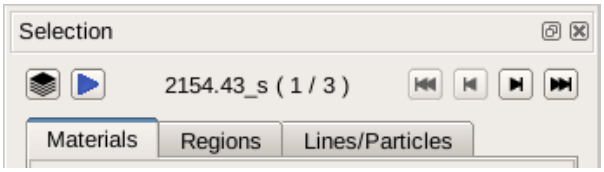


Figure 42 Navigation area for 2D and 3D plots generated from TDR file containing multiple states

The navigation area allows you to switch between displayed states quickly with the:

- Next State button
- Previous State button
- First State button
- Last State button

With any change to the state index, the plot title will be updated to show the current state name.

In addition, the Play button allows you to automatically go through all the states, with a 1 second delay between changes, in ascending order. If the last state is reached and the Play button is still active, the sequence will restart.

The navigation area for 2D and 3D plots also has the Expand/Collapse States button that displays the Expand States dialog box where you can select the states to expand (see Figure 43):

- Click the **>** button to add one state only.
- Click the **<** button to remove one state only.
- Click the **>>** button to add all states.
- Click the **<<** button to remove all states.

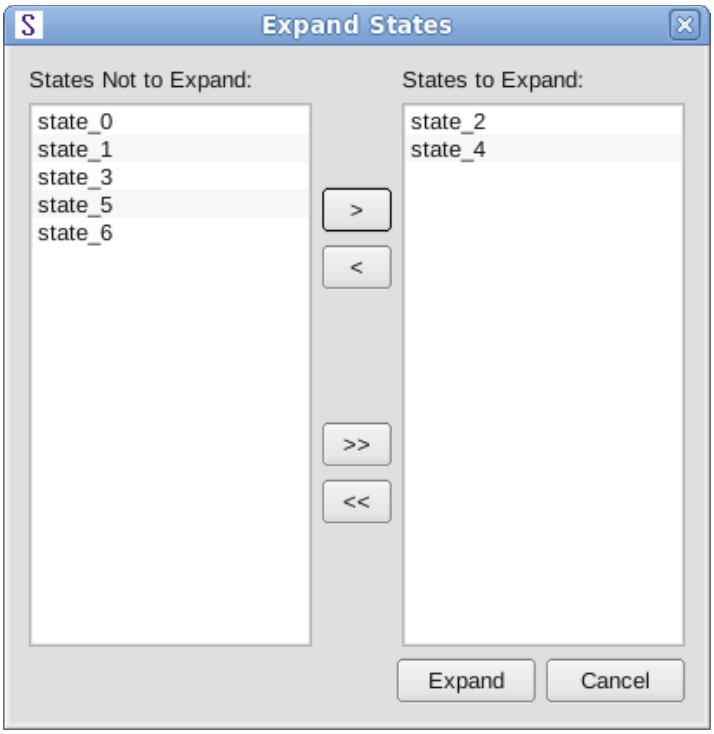


Figure 43 Expand States dialog box

When you click the **Expand** button in the dialog box, all the selected states are expanded to separate plots, so that you can analyze each state one by one (see Figure 44 on page 66).

If you click the Expand/Collapse States button again in any expanded plot or the parent plot, all of the expanded plots will collapse, but not the parent plot.

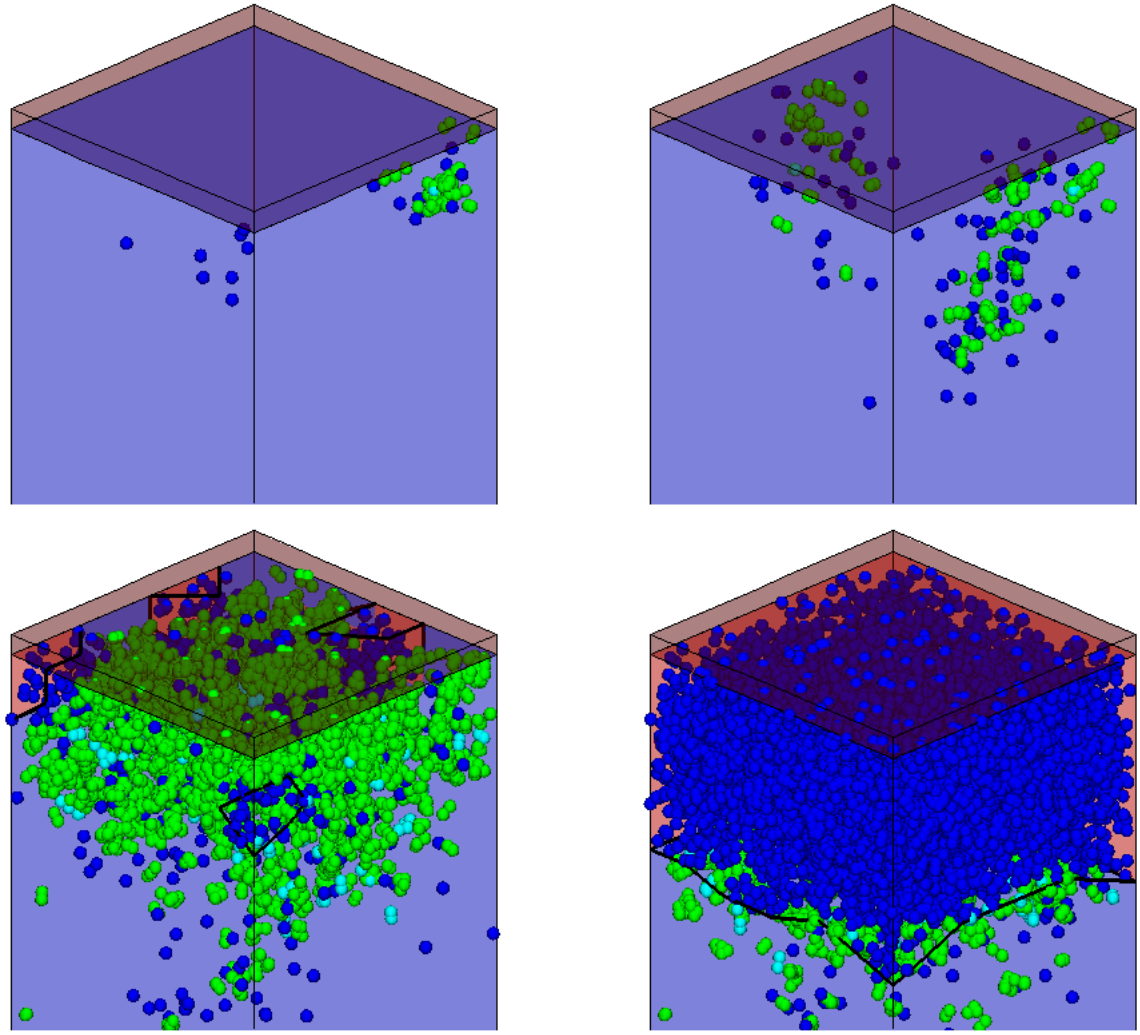


Figure 44 Example of 3D kinetic Monte Carlo TDR file expanded to show the same structure with changes to the state over time

Such plots can also have different field data for each state. Switching states or expanding plots can help you to visualize data (see Figure 45).

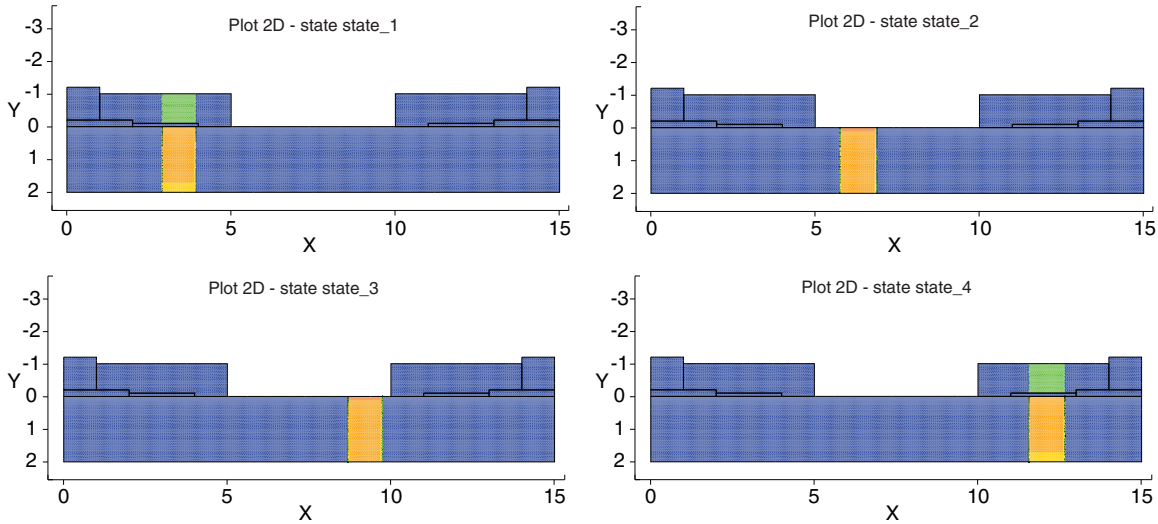


Figure 45 Example of multistate TDR file that has been expanded to show field data as separate plots

3D View

The 3D view is described by the position and the orientation of the camera in the world coordinate system.

The orientation of the camera is described by the vector formed between its position and focal point. This vector is called the *direction of propagation*. Initially, the center of the structure is located at the focal point.

The position can be described in a spherical coordinate system by the distance between the center of the camera and the focal point, also known as the *depth distance*, and two angles (azimuth and elevation).

Figure 46 on page 68 describes the parameters that define the camera position.

The camera has its own coordinate system that is defined by three vectors: the view up, the direction of propagation, and the horizontal vector.

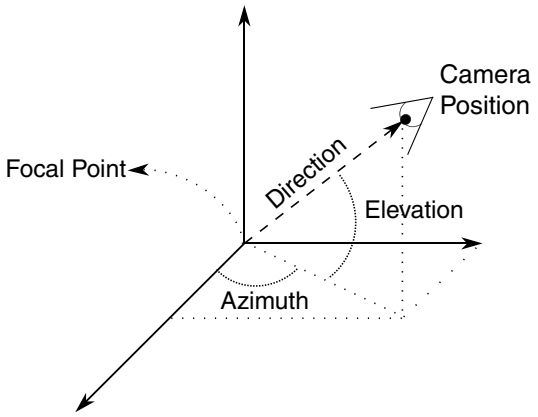


Figure 46 Parameters used to define camera position

The view is defined by the view angle of the camera and the location of the focal plane, which is the plane defined by the focal point and the view up vector. The projection of this plane in the screen will be the view observed. Figure 47 on page 69 shows the variables that describe the camera and the final view given by these variables.

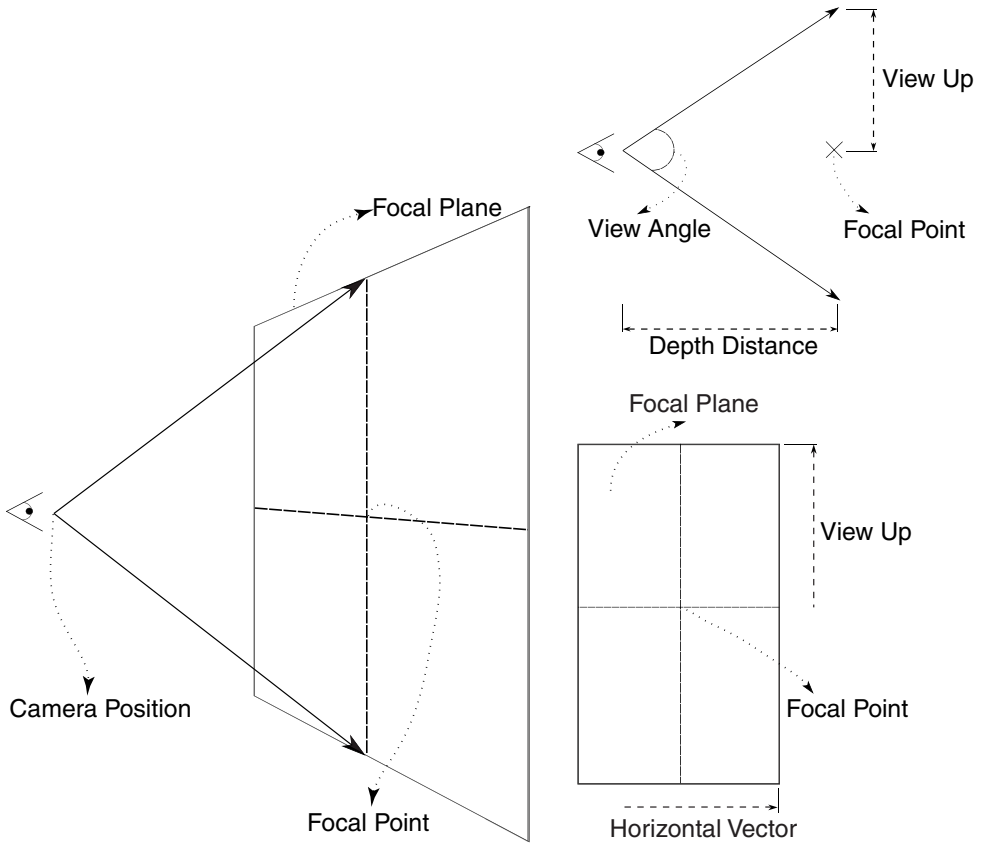


Figure 47 Camera properties: (*left*) perspective view, (*upper right*) horizontal view, and (*lower right*) vertical view

Interacting With 3D Plots

While a 3D plot is active, it is possible to interact directly with the camera.

Using the Select/Rotate tool and dragging, you can rotate the view in relation to the rotation center using the default rotation mode.

Using the Spherical Rotation tool and dragging, you can perform a spherical rotation of the view. Originally, the rotation center is the same as the focal center and can be changed by pressing the O key (lowercase character). The rotation transformation changes the azimuth and the elevation angles (taking the rotation center as reference), thereby maintaining the distance from the origin constant. In addition, you can rotate the plot more accurately in each angle by choosing **View** > **Rotate** or using the `rotate_plot` command. You can specified the use of either the x-, y-, and z-coordinates, or the psi, theta, and alpha spherical coordinates (see rotate_plot on page 255).

In the same way, by right-clicking, you can change the position of the camera and its focal point in the plane perpendicular to the direction of propagation, keeping the direction of propagation and the depth distance constant.

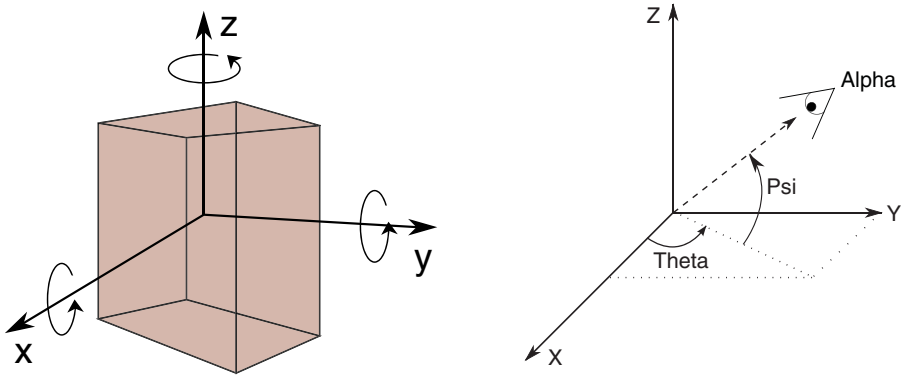


Figure 48 (*Left*) Rotation of axes in relation to rotation center and (*right*) spherical rotation

It is also possible to zoom in to the plot, changing the depth distance using the mouse wheel.

Furthermore, with the `zoom_plot` command, you can specify a factor to change the view angle of the camera (see zoom_plot on page 302). The factor given to the command is multiplied by the *view up* vector (the reflection of the view angle in the focal plane), resulting in zooming in when the factor is greater than 1 and zooming out when it is less than 1.

In addition, the positions of the camera and its focal point, in the world coordinate system, can be changed using the Camera Configuration dialog box (choose **View** > **Camera Configuration**).

2D View

The visualization of a 2D plot is described by the same camera values as the 3D visualization, but the difference is that, in a 2D view, the direction of propagation never changes, thereby maintaining constant azimuth and elevation angles.

Interacting With 2D Plots

Dragging across a plot changes the position of the camera and its focal point in the plane perpendicular to the direction of propagation, which is the same plane for 2D plots.

Using the mouse wheel changes the depth distance from the focal point, zooming in to and zooming out of the plot.

Rendering Options

Two-dimensional or 3D plots are composed of materials that are distributed in regions with properties defined in contour maps (scalars) or flux lines (vectors). All these properties can be found on the Selection panel (see Figure 49).

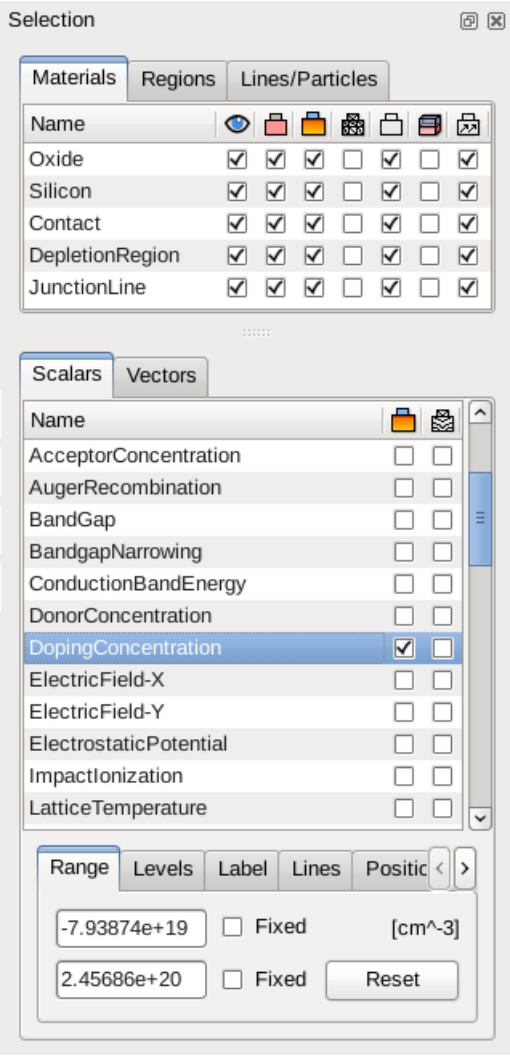


Figure 49 Selection panel showing materials and scalar properties

Materials and Regions

The materials and regions of which a plot is composed are shown in the upper part of Figure 49 on page 71, and Table 5 describes the icons relevant to materials and regions.

NOTE Double-clicking a cell of a structure in the plot area highlights the region or material to which that cell belongs in the Selection panel.

Table 5 Icons for materials and regions

Icon	Description
	Shows or hides the material or region completely. If disabled, it hides the bulk, contour fields, mesh, borders, and vector fields independent of their state.
	Shows or hides the bulk.
	Shows or hides the contour fields.
	Shows or hides the mesh.
	Shows or hides the borders.
	Enables or disables translucency.
	Shows or hides the vector fields.

Showing or Hiding Properties for Multiple Materials and Regions

Clicking a check box next to a material or region shows or hides that specific property only for that region or material.

NOTE You can select multiple rows of materials or regions by dragging the cursor, or holding the Ctrl key or the Shift key while selecting rows in the Selection panel.

If you select multiple rows of materials or regions, when you click an icon itself, it shows or hides that specific property only for all the selected materials or regions.

If no materials or regions are selected, clicking an icon affects all materials or regions. These operations are immediately shown in the plot area.

Modifying Properties in Multiple Materials and Regions

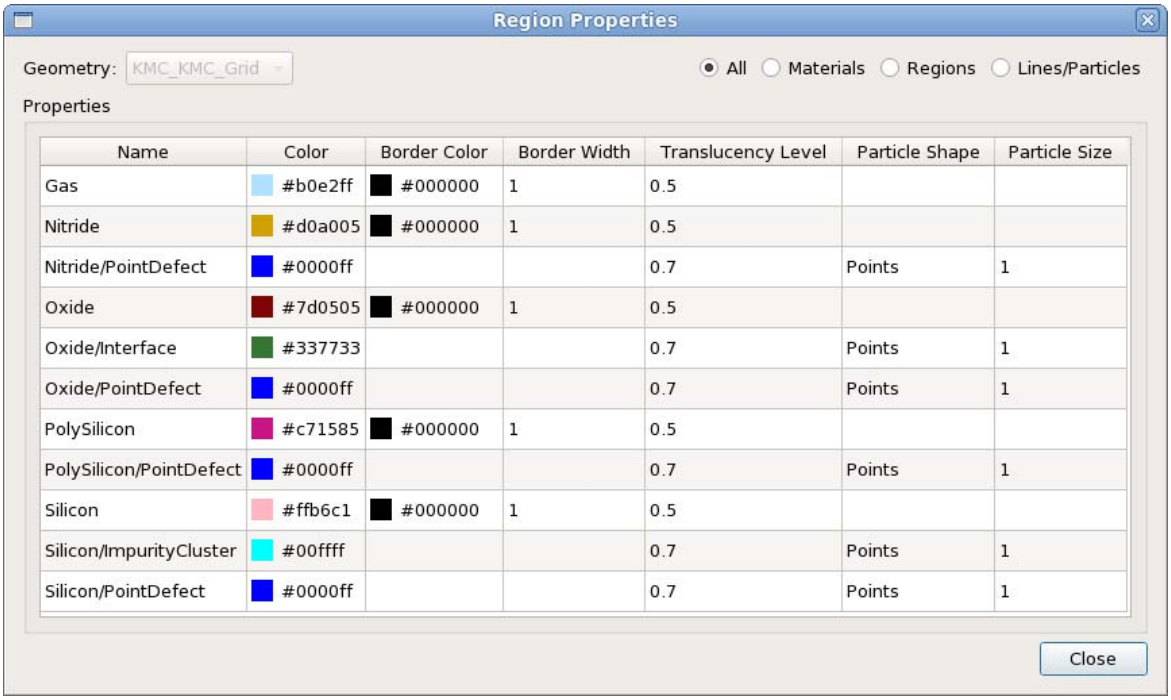
Sentaurus Visual provides a dialog box where you can modify all properties in several regions, particles, or materials at the same time (see Figure 50).

To modify multiple regions:

1. Choose **Data** > **Region Properties**, or click the toolbar button.
2. Select the required rows.
3. Click the column header of the property you want to change.

   A dialog box is displayed where you enter the value of the property.

However, if only one region needs to be modified, double-click the entry and type the new value to change the property.



Name	Color	Border Color	Border Width	Translucency Level	Particle Shape	Particle Size
Gas	#b0e2ff	#000000	1	0.5		
Nitride	#d0a005	#000000	1	0.5		
Nitride/PointDefect	#0000ff			0.7	Points	1
Oxide	#7d0505	#000000	1	0.5		
Oxide/Interface	#337733			0.7	Points	1
Oxide/PointDefect	#0000ff			0.7	Points	1
PolySilicon	#c71585	#000000	1	0.5		
PolySilicon/PointDefect	#0000ff			0.7	Points	1
Silicon	#ffb6c1	#000000	1	0.5		
Silicon/ImpurityCluster	#00ffff			0.7	Points	1
Silicon/PointDefect	#0000ff			0.7	Points	1

Figure 50 Region Properties dialog box

Contact Regions

The contact material and its regions can be colored in a special way defined in the user preferences. This feature allows users to differentiate the contacts for better understanding of the types of region. Any change to the user preferences is applied to the next created plots.

Nevertheless, the changes can be applied to the current plot using the **Contacts** tab of the Plot Properties panel.

In the User Preferences dialog box, select **2D/3D > Contacts**. In the **Contact Color Behavior** box, select from one of the options:

- Constant Color
- List Colors
- Map Colors

The **Constant Color** option loads all the contacts with the same color as a configurable default value. Magenta is the default.

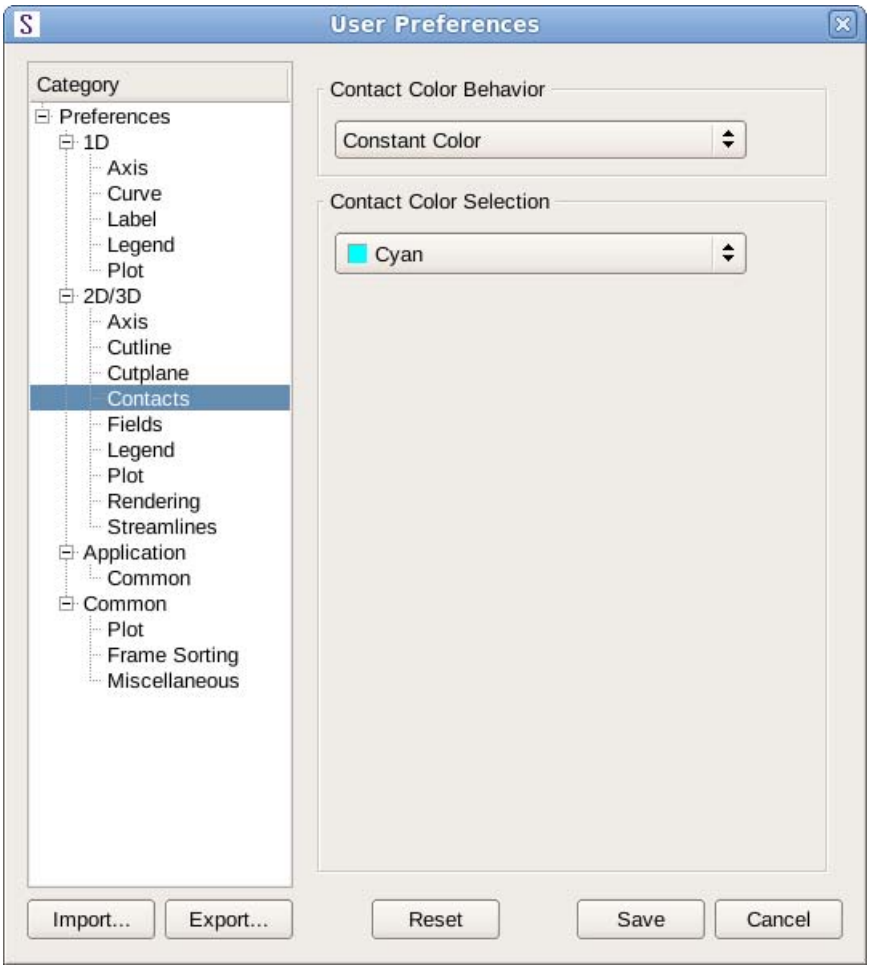


Figure 51 User Preferences dialog box showing selection of Constant Color

The **List Colors** option loads the contacts with a set of colors using round robin logic.

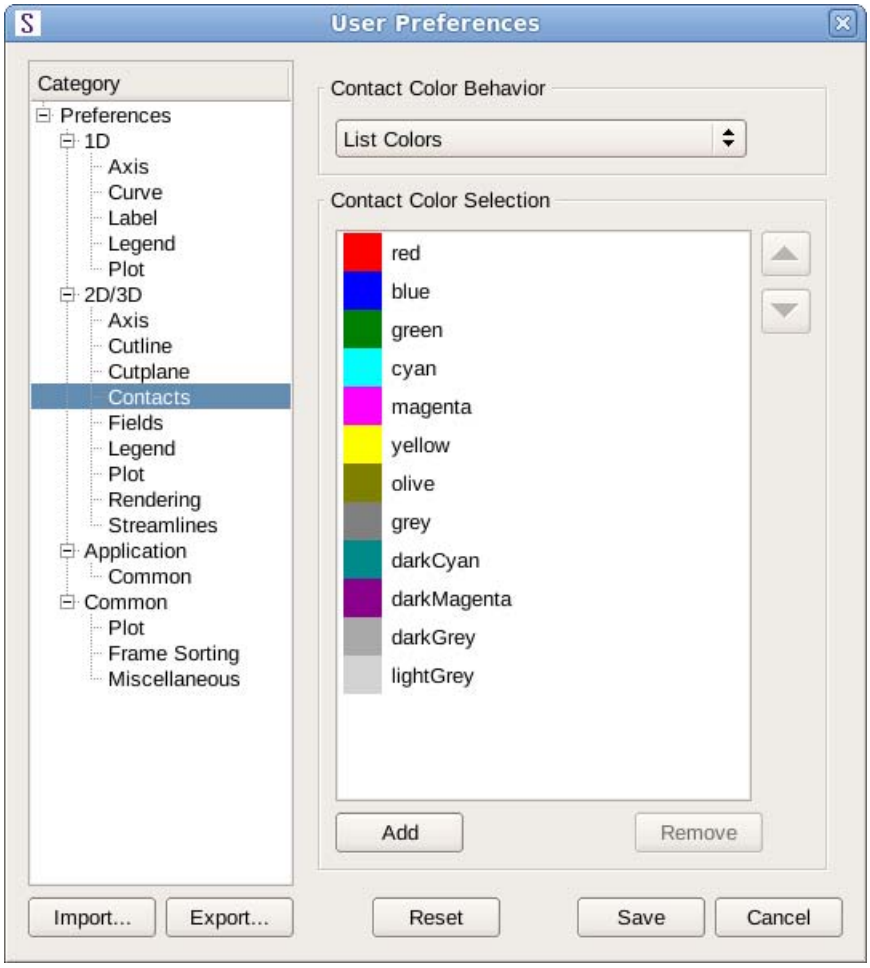


Figure 52 User Preferences dialog box showing selection of List Colors

The **Map Colors** option loads the specified contacts with the specified colors; otherwise, they are displayed with a constant color. The Contact Name column supports wildcards for reference contacts with common patterns in their names. This list is empty by default.

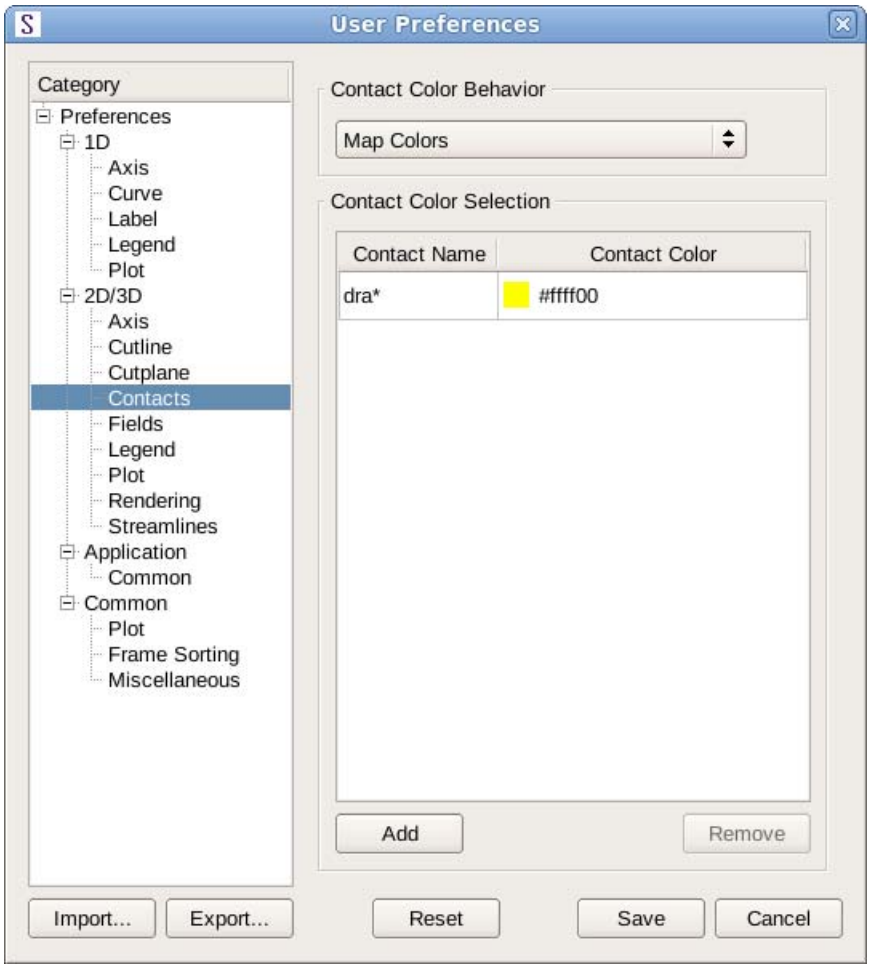


Figure 53 User Preferences dialog box showing selection of Map Colors

To apply changes to the user preferences without reloading a plot, you can use the **Contacts** tab of the Plot Properties panel to select the color behavior and to apply the changes (see Figure 54 on page 77).

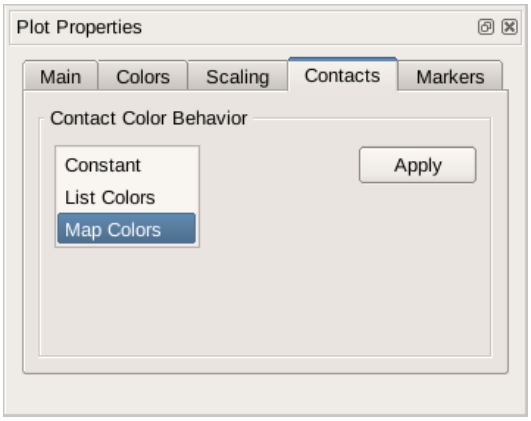


Figure 54 Plot Properties panel showing Contacts tab

Contour Plots

Scalar fields are used to generate contour plots. Usually, the contour levels are calculated automatically, so that they are distributed evenly within the value range of the active field.

Contour Legend Settings

The properties of the legend of contour plots can be changed by double-clicking the legend.

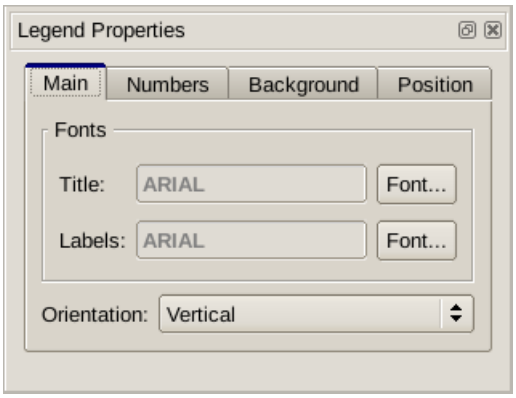


Figure 55 Legend Properties panel

The Legend Properties panel opens (see Figure 55) where you can:

- Customize the number, precision, and notation of the labels.
- Enable a background for the legend.
- Change the background color and the frame color.

- Customize the font for the title and the labels.
- Set the orientation of the legend.

The font size of the legend is related to the diagonal of the plot, and Sentaurus Visual internally sets a value so that the legend is visible in plots that are 600×600 pixels or larger.

To change this value, you must apply a font scale diagonal factor to the font base every time that you rescale a plot. You can do this with the `-title_font_factor` and `-label_font_factor` arguments of the `set_legend_prop` command to change the title and labels of the legend, respectively (see set_legend_prop on page 275).

In addition, you can set this factor in the User Preferences dialog box.

To set a new font scale diagonal factor:

1. Choose **Edit** > **Preferences**.
2. In the User Preferences dialog box, expand **2D/3D** > **Legend**.
3. Under Fonts, click the button next to the **Title Font** or **Label Font** field.
4. In the Font dialog box, set a font scale diagonal factor.

   Values greater than 1.0 will increase the font size, and values less than 1.0 but greater than zero will decrease it.
5. Click **OK** to close the Font dialog box.
6. Click **Save**.

Displaying Contour Plots

To create a contour plot, select the required property to be plotted on the **Scalars** tab of the Selection panel (see Figure 49 on page 71). The range and levels are set automatically, but they can be customized using the **Range** and **Levels** tabs (see Figure 56), where you can manually define a range and the number of levels displayed.

Figure 56 Contour plot options showing the Range tab where the first field is the minimum value and the second field is the maximum value of the range

On the **Lines** tab, you can change the properties of the contour lines of the selected field, such as the style, color, and width, and then show several contour lines at the same time.

All of the default values of field scaling and field units are defined in the `datexcodes.txt` file for each field (see Utilities User Guide, Variables on page 2). In addition, if a field is defined (in the `datexcodes.txt` file) to be present only in one type of material (for example, semiconductor), no data will be loaded (or displayed) for regions of a different material type.

Although only one field can be displayed using color-filled contour levels, you can display multiple contour lines from other fields by clicking the second column of the field list of the Selection panel (see Figure 49 on page 71).

Converting Data to Nodal

For element-type data, there is an option that interpolates element data to nodal data.

To convert data, select the **Convert to Nodal** option on the **Levels** tab of the Selection panel. Figure 57 on page 80 shows the results of converting data to nodal. The converted data looks smoother.

In addition, in the User Preferences dialog box (expand **2D/3D > Plot**), the **Convert Element To Nodal Data** option is selected by default.

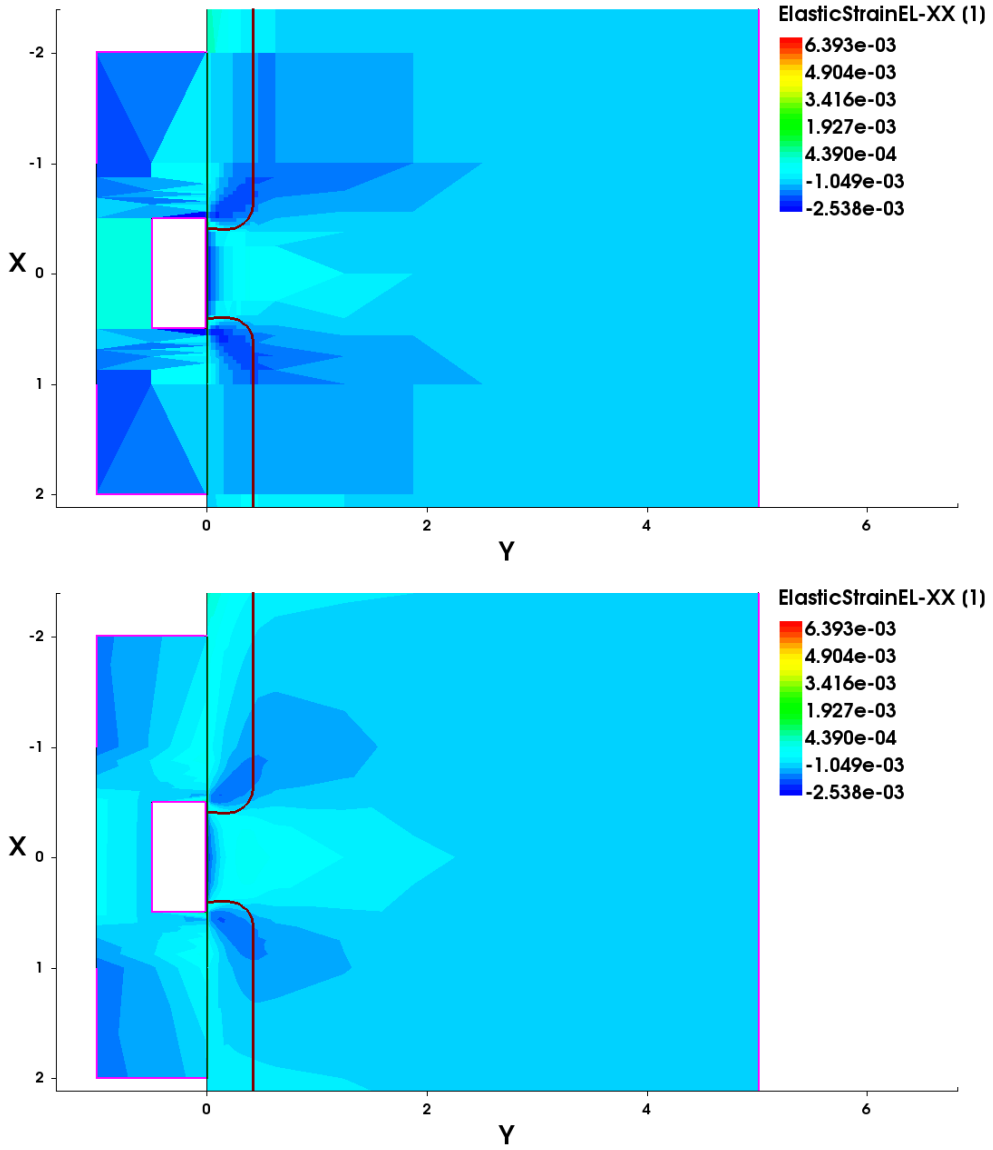


Figure 57 Comparison of (*top*) element-type data and (*bottom*) nodal-type data

Creating New Scalar Fields

Custom scalar fields can be created on the **More** tab (see Figure 56 on page 78). When you click the **Add Field** button on this tab, a dialog box is displayed where you can create a custom field (see Figure 58 on page 81). It allows insertion of functions and operators, and the use of existing fields.

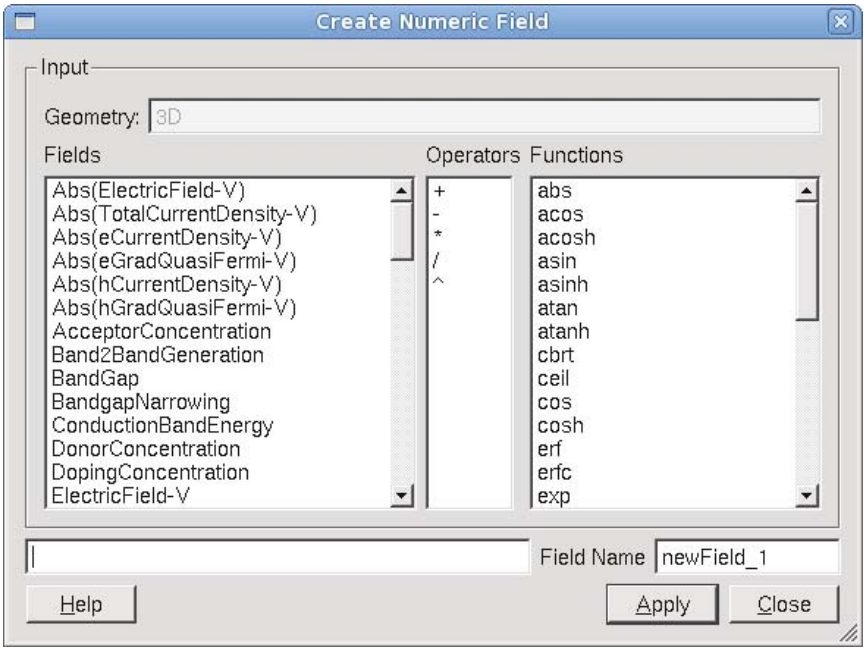


Figure 58 Create Numeric Field dialog box displaying fields, operators, and functions

Vector Plots

To add a vector field to a plot, click the **Vectors** tab of the Selection panel. Select a check box next to a field to display it on the plot. Vector lines can be displayed uniformly or with a size proportional to the magnitude of the field.

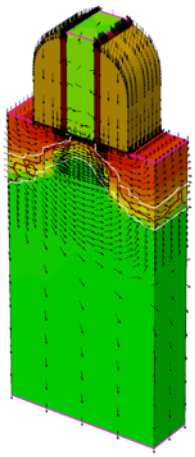


Figure 59 Example of plotting a vector field

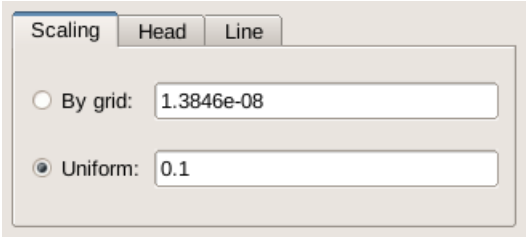


Figure 60 Scaling options for vectors

Uniform scaling means that all vectors have the same size. The length of the arrow is the value of the scaling.

With the grid scaling option, the length of the arrow is given by the vector field magnitude (or the absolute value if required) multiplied by the scaling factor.

The default uniform value can be set in the User Preferences dialog box (select **2D/3D** > **Fields**). In the Vector group box, if no value is set, Sentaurus Visual uses 0.1 as the default.

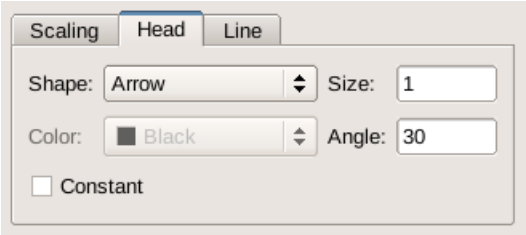


Figure 61 Head tab showing property options for arrowheads

The **Head** tab lists the properties of the arrowhead such as shape, size, angle, and color (only available when the shape selected is **Arrow Solid** or **Head Solid**).

The **Constant** option maintains the size of the arrowhead regardless of the zoom level or the vector length.

Importing an Image as a Background Field

You can load an image file into Sentaurus Visual and use it as a background field.

To load an image into Sentaurus Visual:

1. Choose **File** > **Import Image**.

   The Import Image dialog box is displayed (see Figure 62 on page 83).

2. Click the **Browse** button and select the image, or enter the path to the image in the **File Name** field.
3. In the **Name** field, enter a name for the new dataset containing the image data.
4. Select the **New Plot** option to create a new plot containing the dataset.

   If this option is not selected, the dataset will be overlaid onto the active plot.
5. Click **OK**.

Alternatively, you can load an image using the `load_file` and `overlay_plots` Tcl commands, for example:

```
% load_file "pic.png"
% overlay_plots {Plot_2D} -datasets {pic}
```

See load_file on page 236 and overlay_plots on page 241.

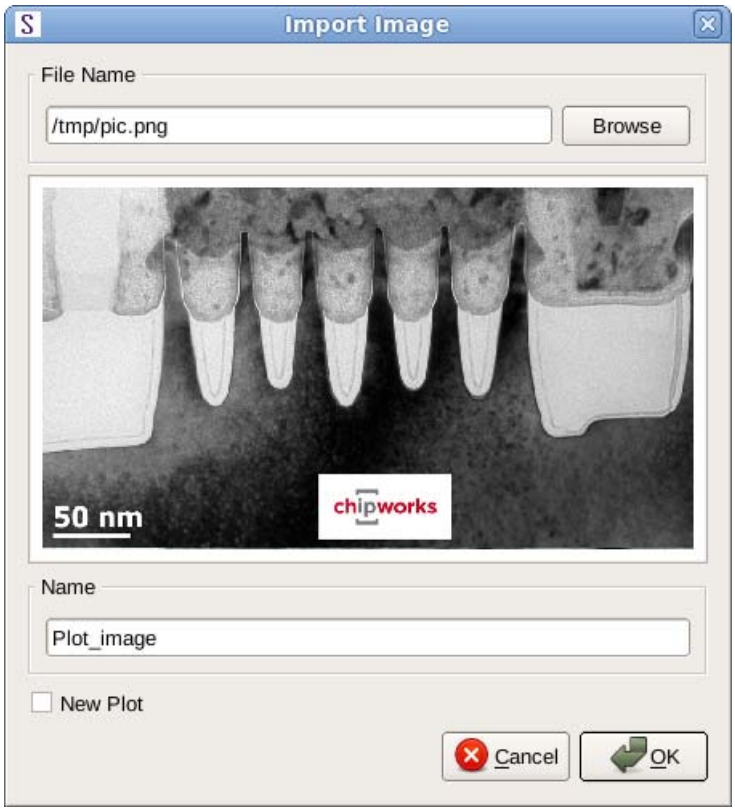


Figure 62 Import Image dialog box

After an image is loaded and overlaid onto a plot, you can adjust the magnification to a specific area in the image using the Scale to Image dialog box (choose **View** > **Scale to Image**).

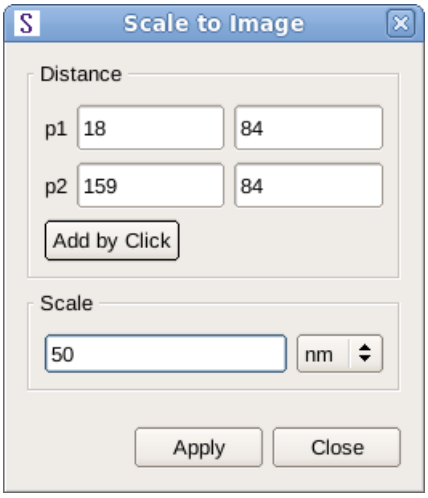


Figure 63 Scale to Image dialog box

To adjust the zoom in the image:

1. In the Scale to Image dialog box, specify two points of the screen – the first is the x-coordinate and the second is the y-coordinate.
2. In the **Scale** field, specify the length and unit that will be the scale used for the distance between the two points.
3. Click the **Apply** button.

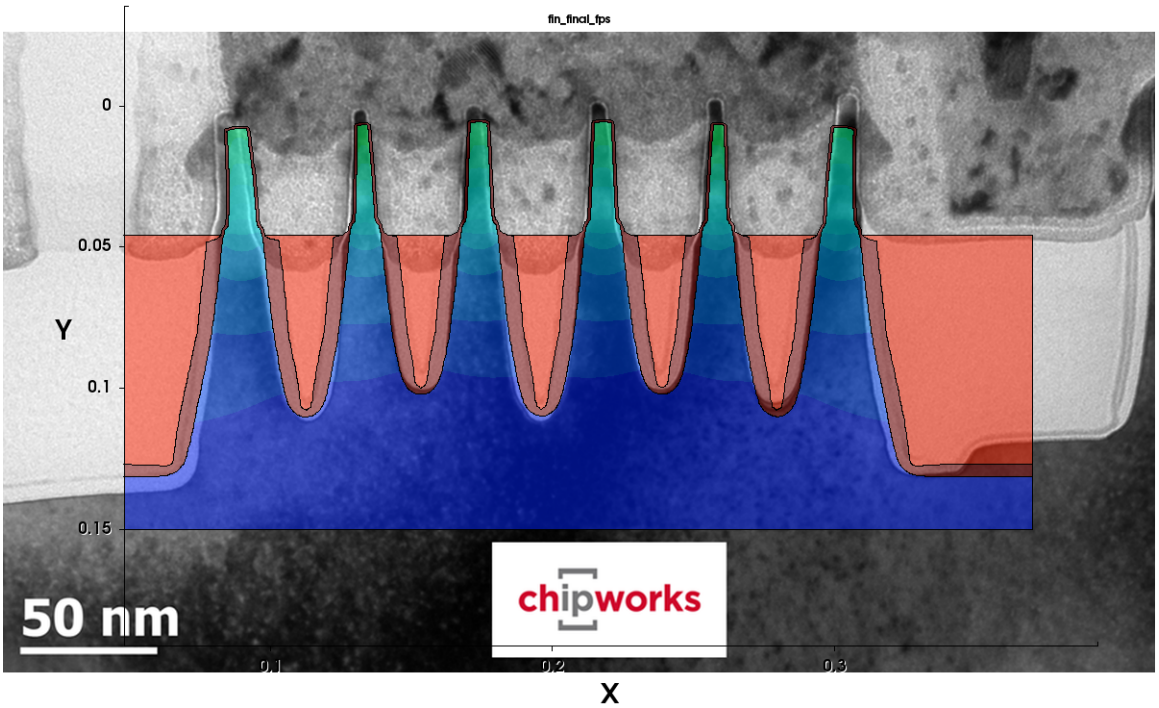


Figure 64 Example of a plot with an overlaid image

Scaling and Shifting 2D and 3D Geometries

Sentaurus Visual can transform the geometry of 2D and 3D geometries, and change how they are visualized in a plot. Two geometric transformations are available: scale and shift.

To scale or shift a geometry, use the Transformation dialog box (choose **Tools** > **Transformation**).

In the Transformation dialog box, the scale and shift values for each axis can be changed for a certain geometry. Click the **Apply** button to apply the changes to the geometry.

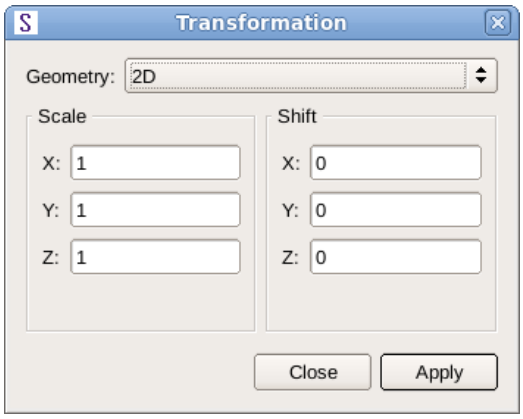


Figure 65 Transformation dialog box

Rotating Structures (3D Plots Only)

Three-dimensional plots can be rotated freely over a rotation point or fixed to an axis.

To rotate a plot, you must be in selection mode (click the toolbar button). Drag to rotate the plot. When you release the mouse button, the rotation stops.

To set the origin of a rotation point, use the camera configuration by either choosing **View** > **Camera Configuration** or placing the cursor on the required point and pressing the O key.

Table 6 Rotation modes

Toolbar button	Shortcut keys	Description
	Press the N key while dragging the cursor.	Enables standard rotation until you release the N key.
	Press the S key while dragging the cursor	Enables spherical rotation until you release the S key.
	Press the X key while dragging the cursor	Fixes the rotation around the x-axis until you release the X key.
	Press the Y key while dragging the cursor	Fixes the rotation around the y-axis until you release the Y key.
	Press the Z key while dragging the cursor	Fixes the rotation around the z-axis until you release the Z key.

Rotation Point

The rotation point is the reference location for free rotation, that is, when not rotating around an axis. Default coordinates are calculated for every plot, locating the rotation point in the center of the initial visible structure. The same rotation point also is used in Spherical Rotation mode.

The rotation-point coordinates can be changed along with the properties of the rotation point by using either:

- The menu bar (choose **View** > **Camera Configuration** and, in the Camera Properties panel, click the **Rotation Point** tab)
- The `set_camera_prop` command (see set_camera_prop on page 264)

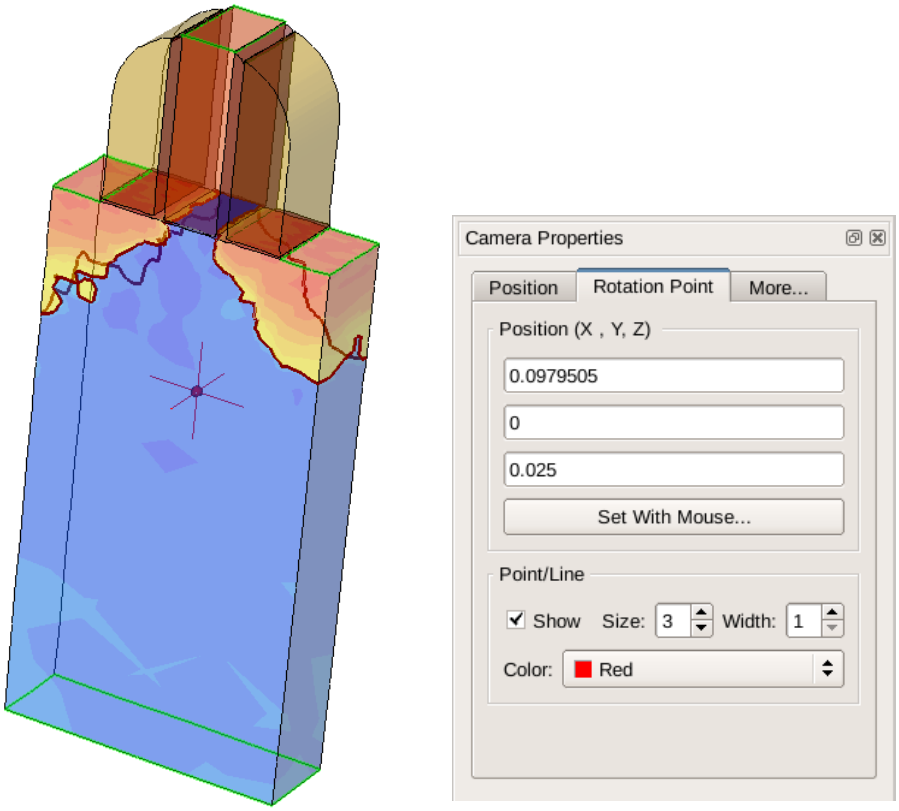


Figure 66 (*Left*) Three-dimensional structure showing the default rotation point (translucency activated) and (*right*) the Rotation Point tab in the Camera Properties panel

Customizing the Rotation Point

The rotation point is visualized as a tiny sphere with axes crossing it over three dimensions. Moreover, the properties of the rotation point can be customized. Table 7 summarizes these properties.

Table 7 Properties of the rotation point that can be customized

Property	Description	Tcl command example
Color	Determines the color of the rotation point when it is visible in `<#rrggbb>` format.	`set_camera_prop -rot_color #FF00FF`
Size	Sets the length of the rotation point axes. The size is an integer.	`set_camera_prop -rot_size 5`
Width	Sets the thickness of the rotation point axes. The width is an integer.	`set_camera_prop -rot_width 3`
Visibility	Determines whether to show or hide the rotation point.	`set_camera_prop -show_rotation_point` `set_camera_prop -hide_rotation_point`
Position	Sets the location of the rotation point. The position is defined by three floating-point values.	`set_camera_prop -rotation_point {0.0 1.0 -0.8}`

Using the Rotation Point as a Reference Point

The rotation point can be used as a reference point when inspecting 3D structures. As previously mentioned, you can move the rotation point to a specific position by either using the GUI or using the `set_camera_prop` command. Using the Tcl command, it is only possible to set the position by exact coordinates, for example:

```
set_camera_prop -rotation_point {0.0 1.0 -0.8}
```

Using the GUI, the position of the rotation point can be set by either exact coordinates or the cursor. In both cases, you must show the **Rotation Point** tab by choosing **View > Camera Configuration** to display the Camera Properties panel. On the **Rotation Point** tab, the **Position** fields allow you to introduce the values of the x-axis, y-axis, and z-axis of the position. The position is updated every time you leave any of the fields or if you press the Return key.

Another option is to use the **Set With Mouse** button. To set the position of the rotation point with this feature:

1. Click the **Set With Mouse** button.

   The button remains selected. Note that the cursor changes appearance to a cross when it hovers over the plot.

2. Click the structure at the required location to set the position of the rotation point. Note that the point will be set at the surface of the structure.

After this, the **Set With Mouse** button is released, and the cursor returns to its previous state.

The rotation point is visible while the **Set With Mouse** button remains selected. Since the rotation point is in the inner part of the structure by default, it is not visible unless translucency is enabled or any region obstructing its view is hidden.

To make the rotation point permanently visible, select the **Show** option (see Figure 66 on page 87 (*right*)). Clear the option to hide the rotation point.

NOTE A shortcut exists to set the rotation point when using the **Set With Mouse** button: Hover the cursor in the required location over the structure and press the O key. This will set the position of the rotation point in the same way as the **Set With Mouse** button.

To set the rotation point inside the structure, you can either:

- Set the rotation point on the surface and, then modify it by using the `set_camera_prop` command or by setting the values in the **Position** fields.
- Hide regions or materials before setting the rotation point with the cursor.

NOTE The position of the rotation point is constant with regard to deformation, value blanking, and other Sentaurus Visual features that alter the structure.

Rotating Plots Using Exact Values

You can rotate 3D plots precisely using the Rotate dialog box (choose **View > Rotate**) or the corresponding `rotate_plot` command (see rotate_plot on page 255).

The dialog box has the following tabs (see Figure 67 on page 90):

- On the **XYZ Axis** tab, you can rotate the structure around the x-, y-, or z-axis using relative angles with a defined number of steps, or you can set absolute angles.

- On the **Spherical** tab, you can rotate the structure using spherical coordinates (see 3D View on page 67 and Figure 48 on page 70) with a defined number of steps, or you can change the angles directly.

The **Step** field applies to both tabs and is used to define the number of steps.

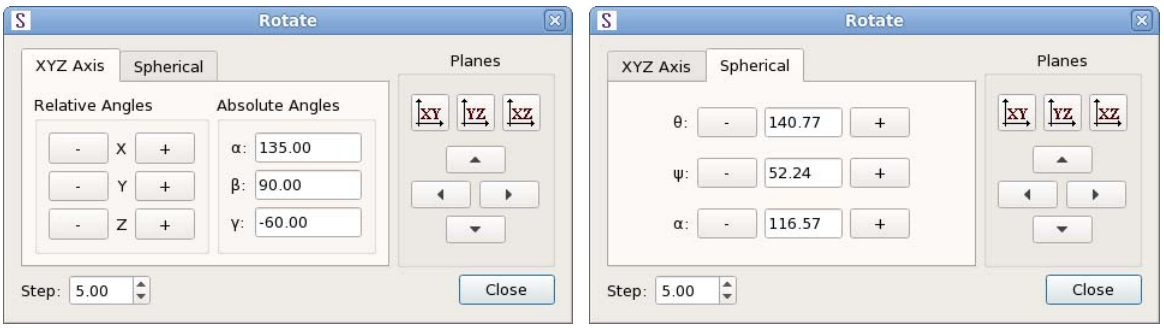


Figure 67 Rotate dialog box showing (*left*) XYZ Axis tab and (*right*) Spherical tab

The Planes group area is independent of the rotation mode. You can change the view of the structure to a specific plane using the **View Plane XY**, **View Plane YZ**, and **View Plane XZ** buttons, or you can rotate the structure 90° in different directions using the arrow buttons.

The rotations performed by the arrow buttons are equivalent to the rotations performed by mouse operations when the rotation mode is in its default state (see Figure 68):

- The **Left Arrow** button and the **Right Arrow** button rotate the plot around an imaginary, completely vertical vector that is located at the rotation point of the plot.
- The **Up Arrow** button and the **Down Arrow** button rotate the plot around an imaginary, completely horizontal vector (perpendicular to the vertical vector) that is located at the rotation point of the plot.

You can use the arrow keys of the keyboard as shortcut keys to rotate the plot in the same way as the arrow buttons of the Rotate dialog box. This functionality is available when a 3D plot is selected or when the Rotate dialog box is open.

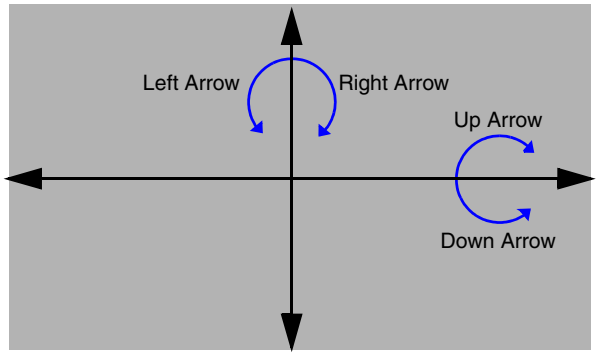


Figure 68 Directions of rotation performed when using the arrow buttons

Overlaying Plots

Overlaying 2D or 3D plots allows you to examine the differences between two similar plots.

To overlay plots:

1. Select two or more plots to be overlaid.
2. Click the  toolbar button.

A new plot is generated with the selected plot structures overlaid.

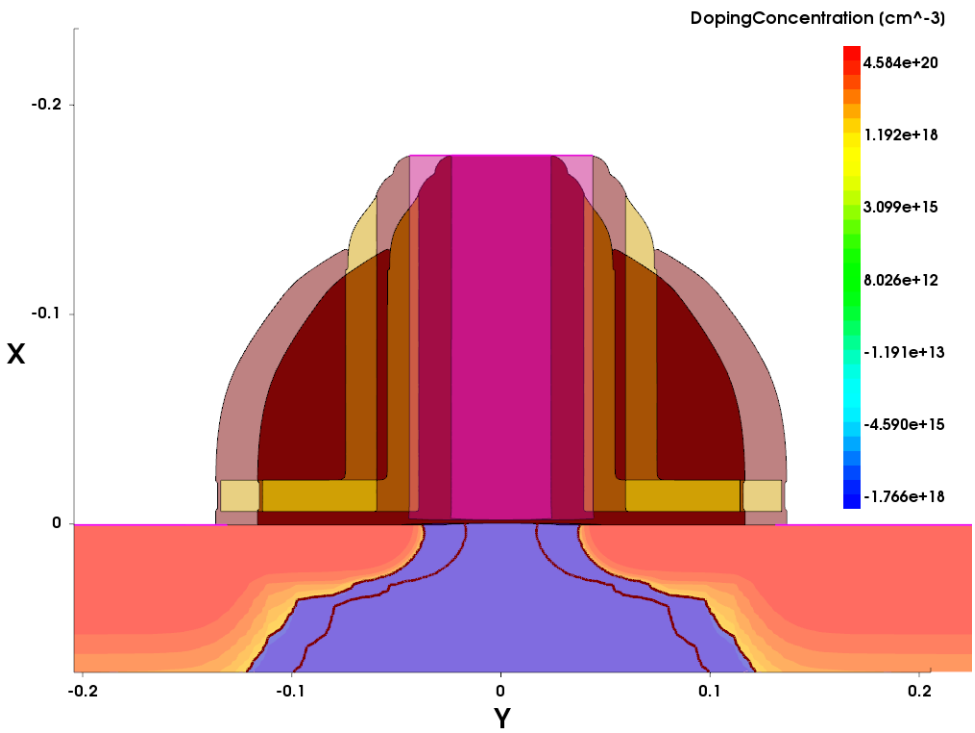


Figure 69 Example of overlaying plots

When plots are overlaid, the Selection panel changes to a tree view to allow for the visualization of different geometries as shown in Figure 70 on page 92.

The geometries can be easily distinguished by selecting different boundary or contour line colors.

To select a specific boundary line color:

1. On the **Materials** tab, double-click the filled rectangle preceding the geometry name.
2. Choose a color from the list, and press the Enter key.

To select a specific contour line color:

1. On the **Scalars** tab, double-click the filled rectangle preceding the geometry name.
2. Choose a color from the list, and press the Enter key.

If you want to change the colors of specific materials or regions, you must use the Region Properties dialog box (press Ctrl+Shift+E). See Modifying Properties in Multiple Materials and Regions on page 73.

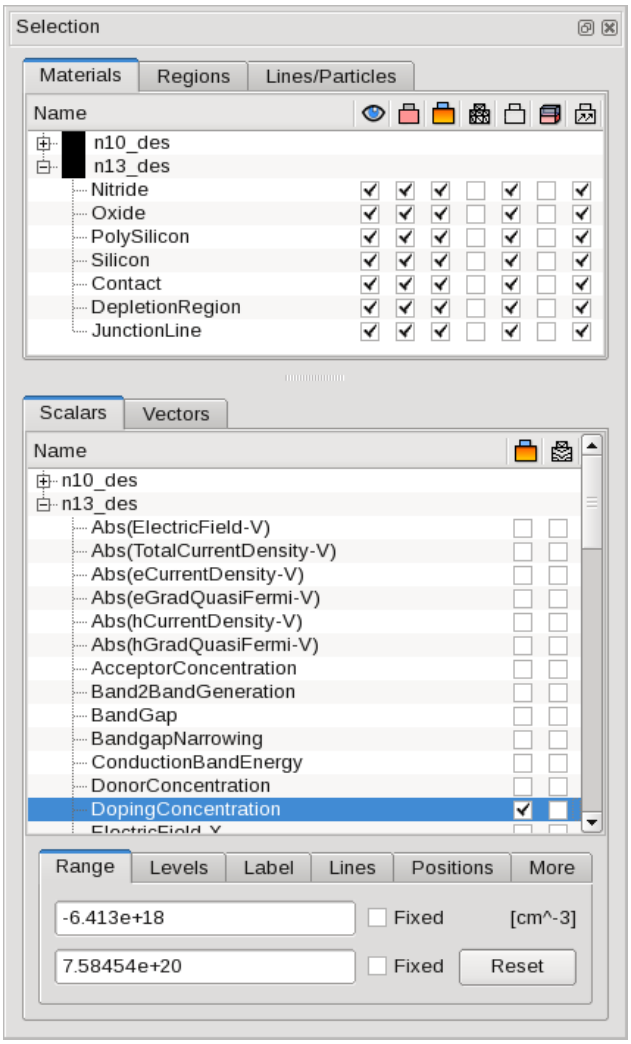


Figure 70 Selection panel showing tree view when plots are overlaid

Showing Differences Between Plots

Differentiating 2D or 3D plots allows you to determine the differences in the common fields of two different plots. A new plot is generated showing the field differences between the two plots.

To differentiate plots:

1. Select two plots with common fields.
2. Click the  toolbar button.

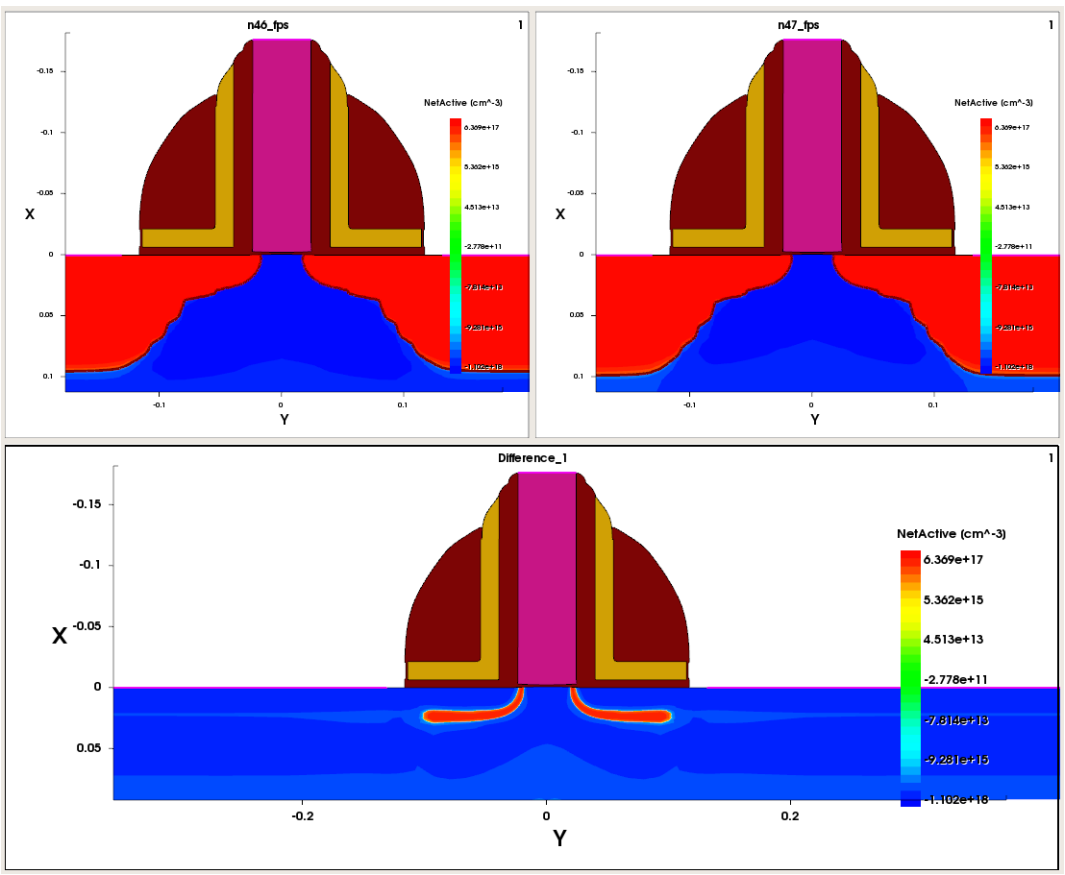


Figure 71 Example of a difference plot resulting from comparison of two plots

Measuring Distances

You can measure the distance between two points in a plane or in space for 2D and 3D plots.

To measure a distance in a plot:

1. Click the **Ruler** toolbar button.
2. Drag from the starting point of the measurement.
3. Release the mouse at the end point of the measurement.

The **Data** tab of the Ruler panel shows the coordinates and distances calculated (see Figure 72).

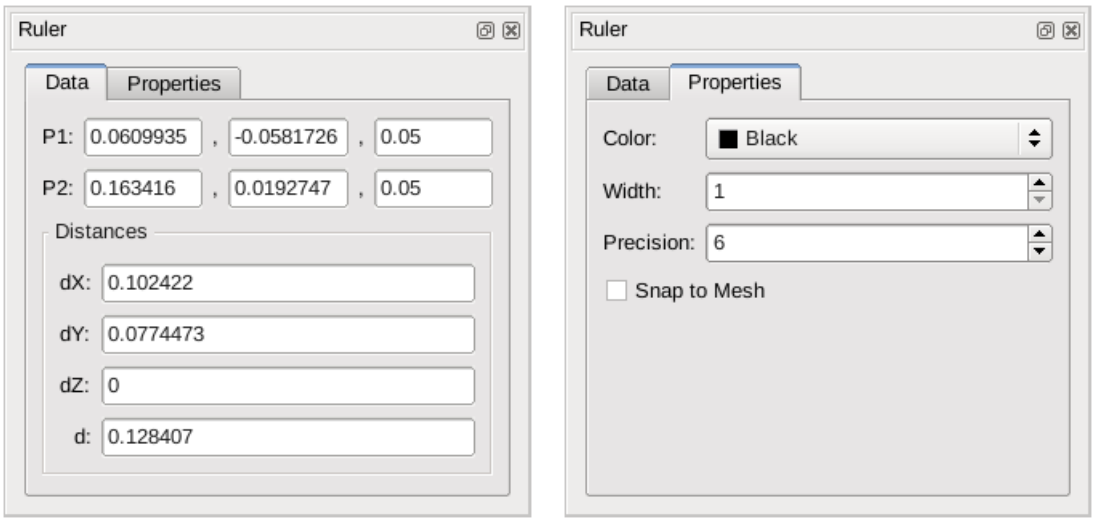


Figure 72 Ruler panel showing (*left*) Data tab and (*right*) Properties tab

The **Properties** tab shows the properties of the ruler and provides a snap-to-nearest point function. When you select the **Snap to Mesh** option, Sentaurus Visual automatically selects the nearest point in the grid to the one clicked when measuring distances.

You can also change the ruler properties using set_ruler_prop on page 286.

While measuring distances, you can rotate a structure using any of the rotation shortcut keys (see Rotating Structures (3D Plots Only) on page 86).

Integration Tool

You can integrate the active field on all the materials of the current 2D or 3D plot.

To enable the integration tool, click the $\int dr$ toolbar button.

The Field Integration dialog box is displayed (see Figure 73) with the results of the integration for each material and a total value calculated over the active field. Integration can be performed on other fields without changing the active field displayed on the structure.

Integration over the active field commences immediately, but it can be stopped by clicking the **Cancel Integration** button of the Field Integration dialog box.

Integration on large structures can take some time. To see the progress of the integration, a progress bar is visible in the lower-right corner of the GUI.

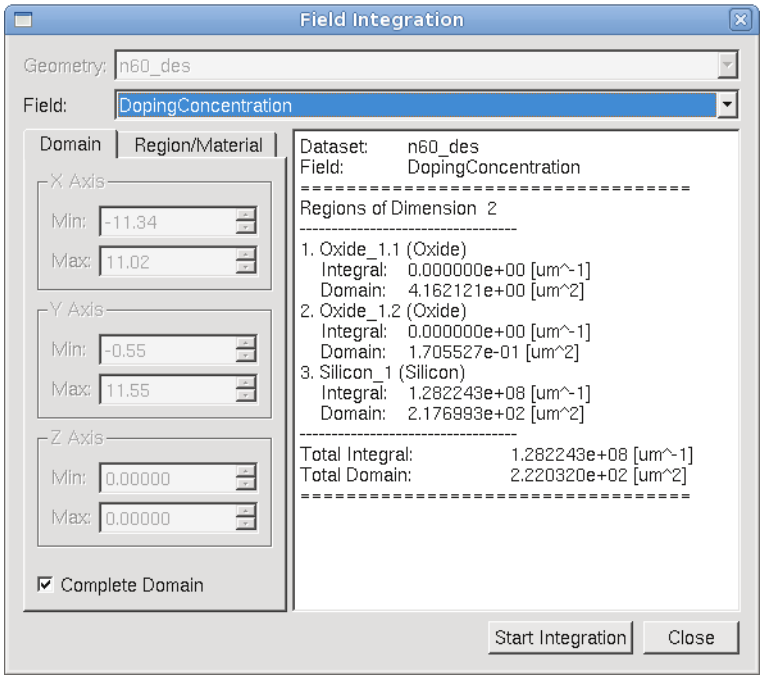


Figure 73 Field Integration dialog box

Using a Custom Integration Domain

Integration can be performed over the complete structure or inside a defined bounding box.

To change the integration domain:

1. In the Field Integration dialog box, clear the **Complete Domain** option.
2. Enter the custom ranges.
3. Click **Start Integration** to obtain the updated value.

Integrating Only a Defined Set of Regions or Materials

By default, integration is performed over all regions or materials of a structure. This can be changed on the **Region/Material** tab of the Field Integration dialog box by selecting only the regions or materials that you want to integrate, and then clicking **Start Integration**.

Probe Tool

The probe tool for 2D and 3D plots allows you to display information about a selected point on a structure.

To probe a point:

1. Click the  toolbar button.
2. Click the point to be evaluated.

The Probe panel opens, which shows various information about the point such as the values of all fields and information about the cell (see Figure 74 on page 97).

NOTE If you hold the Ctrl key when you click the point to be evaluated, the cursor (crosshairs) snaps to the closest mesh point. The same is achieved by selecting the **Snap to Mesh** option on the Probe panel, which provides information about the closest edge to the probed point of the structure.

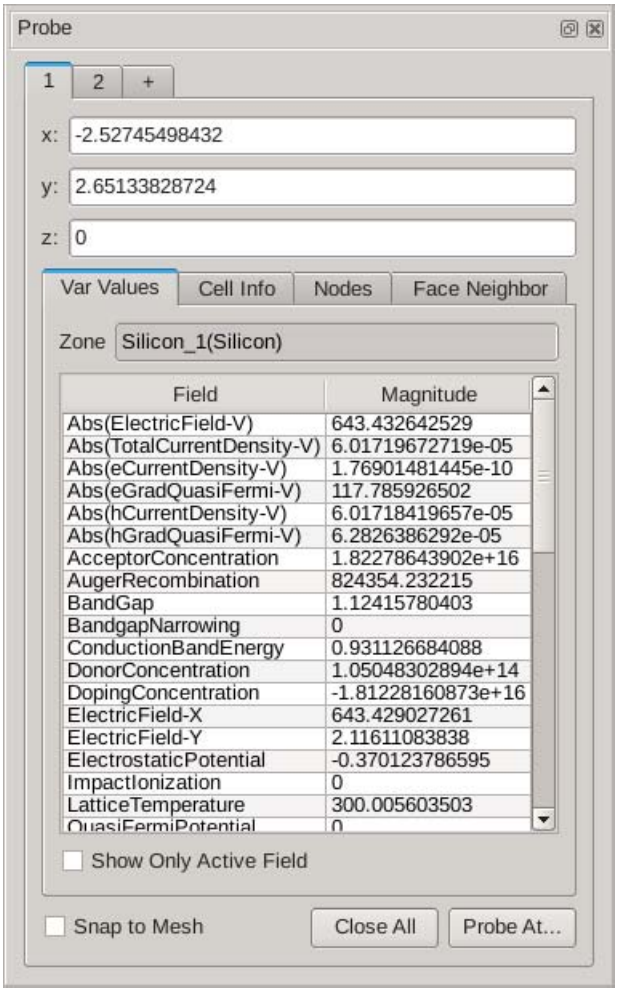
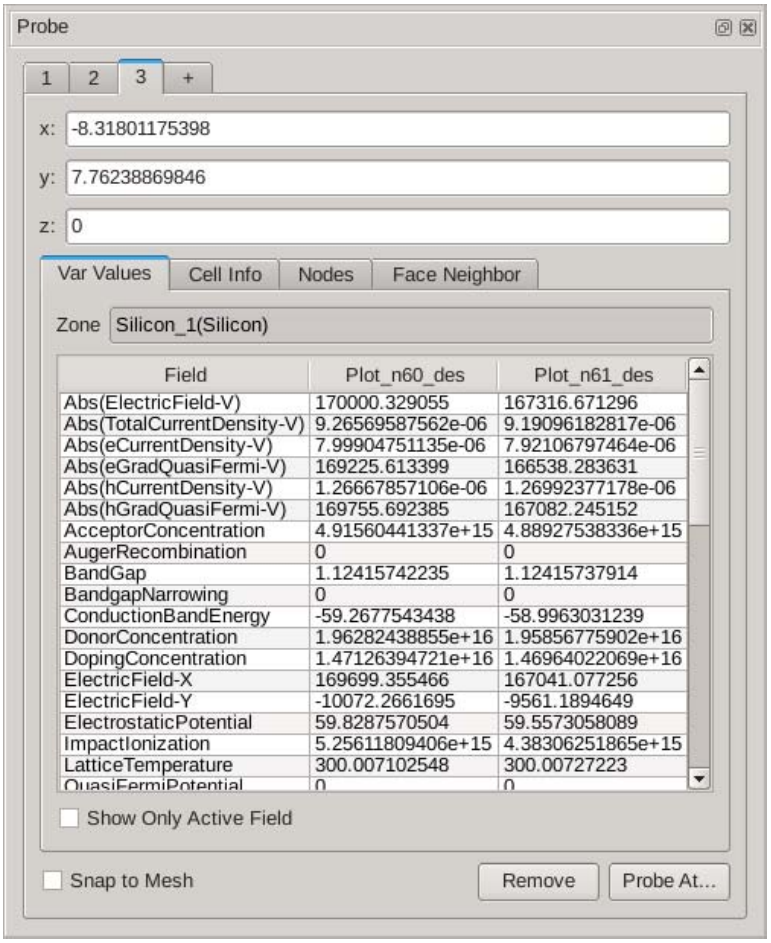


Figure 74 Probe panel

You can keep a selected point on screen and probe other points. Each point has its own numbered tab and you can click tabs to switch the active point and highlight the cursor (crosshairs) in the plot. In the Probe panel, you can click the **Close All** button on the first point tab to delete all points except the first one.

In addition, you can display information from different plot groups at the same time. However, only information that is available for all group members will be shown (see Figure 75 on page 98). If one member of the plot group does not have the same information as the plot group leader, it will show *nan* (not a number) as the current value. This feature is compatible with displaying multiple crosshairs (see Figure 76 on page 99).



Field	Plot_n60_des	Plot_n61_des
Abs(ElectricField-V)	170000.329055	167316.671296
Abs(TotalCurrentDensity-V)	9.26569587562e-06	9.19096182817e-06
Abs(eCurrentDensity-V)	7.99904751135e-06	7.92106797464e-06
Abs(eGradQuasiFermi-V)	169225.613399	166538.283631
Abs(hCurrentDensity-V)	1.26667857106e-06	1.26992377178e-06
Abs(hGradQuasiFermi-V)	169755.692385	167082.245152
AcceptorConcentration	4.91560441337e+15	4.88927538336e+15
AugerRecombination	0	0
BandGap	1.12415742235	1.12415737914
BandgapNarrowing	0	0
ConductionBandEnergy	-59.2677543438	-58.9963031239
DonorConcentration	1.96282438855e+16	1.95856775902e+16
DopingConcentration	1.47126394721e+16	1.46964022069e+16
ElectricField-X	169699.355466	167041.077256
ElectricField-Y	-10072.2661695	-9561.1894649
ElectrostaticPotential	59.8287570504	59.5573058089
ImpactIonization	5.25611809406e+15	4.38306251865e+15
LatticeTemperature	300.007102548	300.00727223
QuasiFermiPotential	0	0

Figure 75 Probe panel showing multiple probe points on linked plots

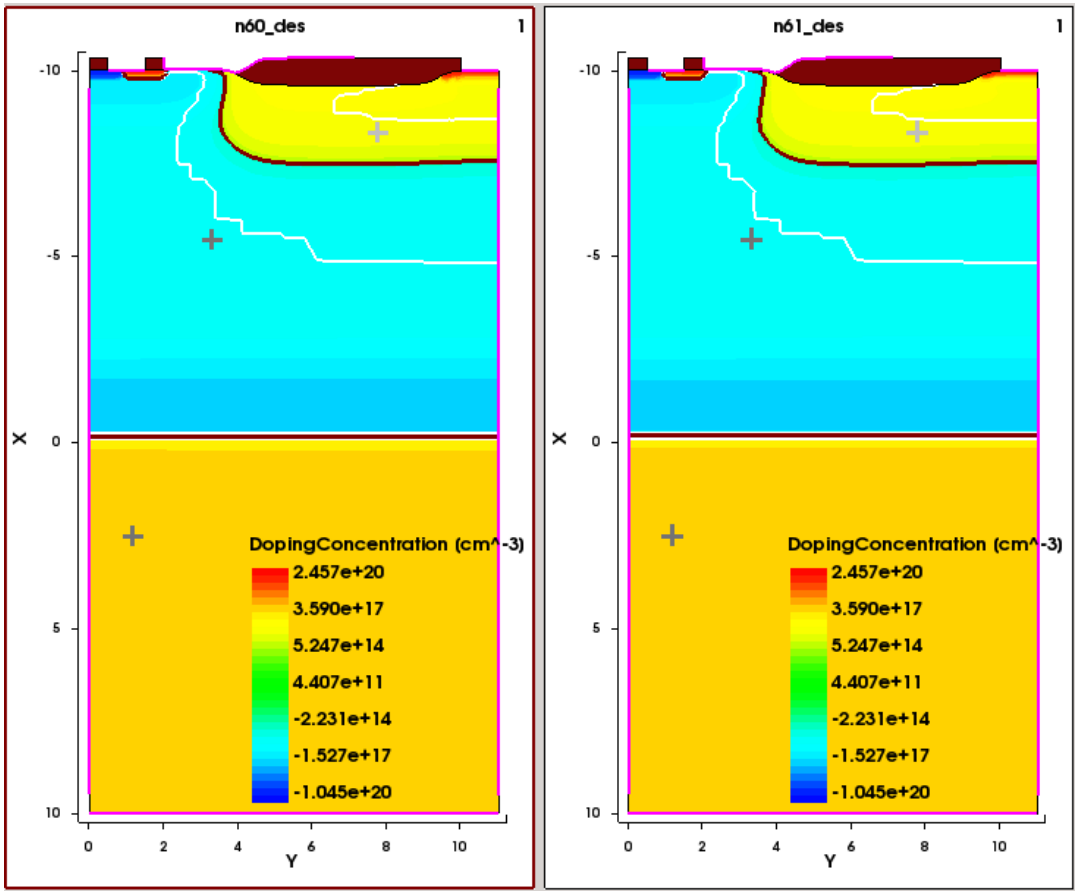


Figure 76 Multiple crosshairs shown in different plot groups

Dataset Information Tool

You can access 2D or 3D dataset information, such as the number of points or elements in a specific material or region with the Dataset Information tool. To display the relevant dialog box, choose **Data** > **Dataset Information**.

In the Dataset Information dialog box, relevant information about the dataset will be displayed, depending of the dataset and the materials or regions selected. In the Datasets group box (see Figure 77 on page 100), all the 2D and 3D datasets are displayed so you can choose the dataset from which the data will be extracted. The number of fields, elements, and points are shown in the table located in the upper-right part of the dialog box.

In the Materials/Regions group box, you can select various materials or regions, and the sum of points and elements of each region selected is displayed in the table in the lower-right part of the dialog box.

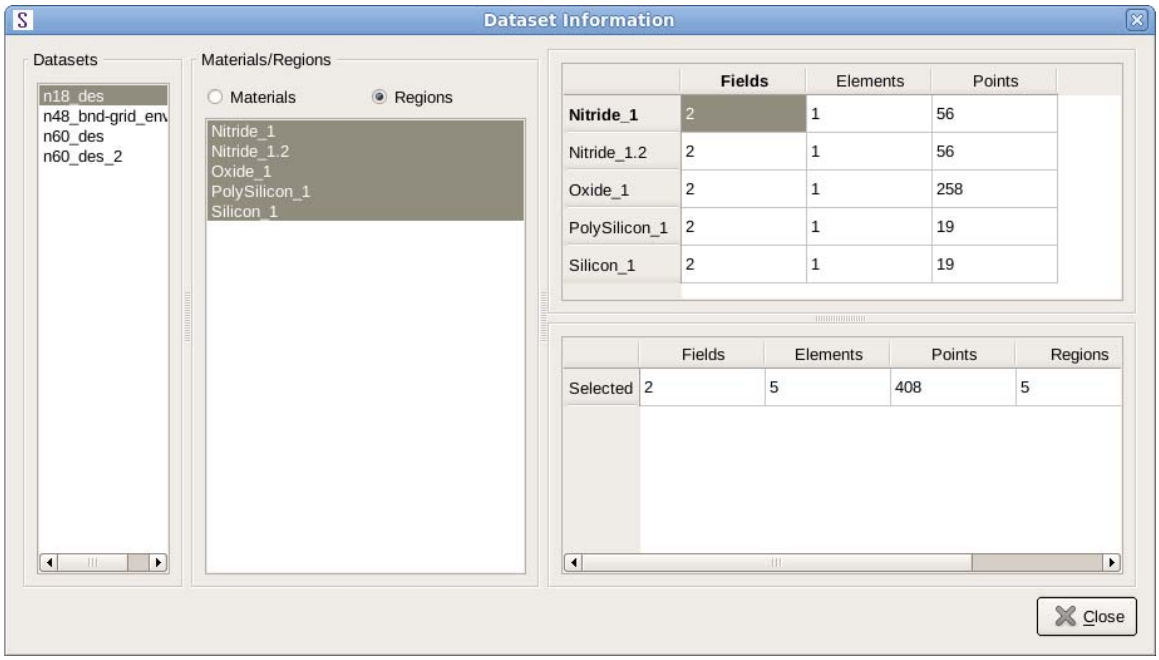


Figure 77 Dataset Information dialog box

The total number of points and elements of a 2D or 3D dataset is displayed at any time at the bottom of the main window. It is important to mention that this value may not be the same at the one displayed in this dialog box, since this value is extracted directly from the geometry and the dialog box sums the points and elements of each region separately.

Maximum and Minimum Locations of Fields

Sentaurus Visual can easily display the maximum and minimum locations of a particular field.

To display these values:

1. On the Plot Properties panel, click the **Markers** tab.
2. Select the **Show Min** option, or the **Show Max** option, or both options.

When either option is selected, a marker like the one shown in Figure 78 on page 101 is displayed.

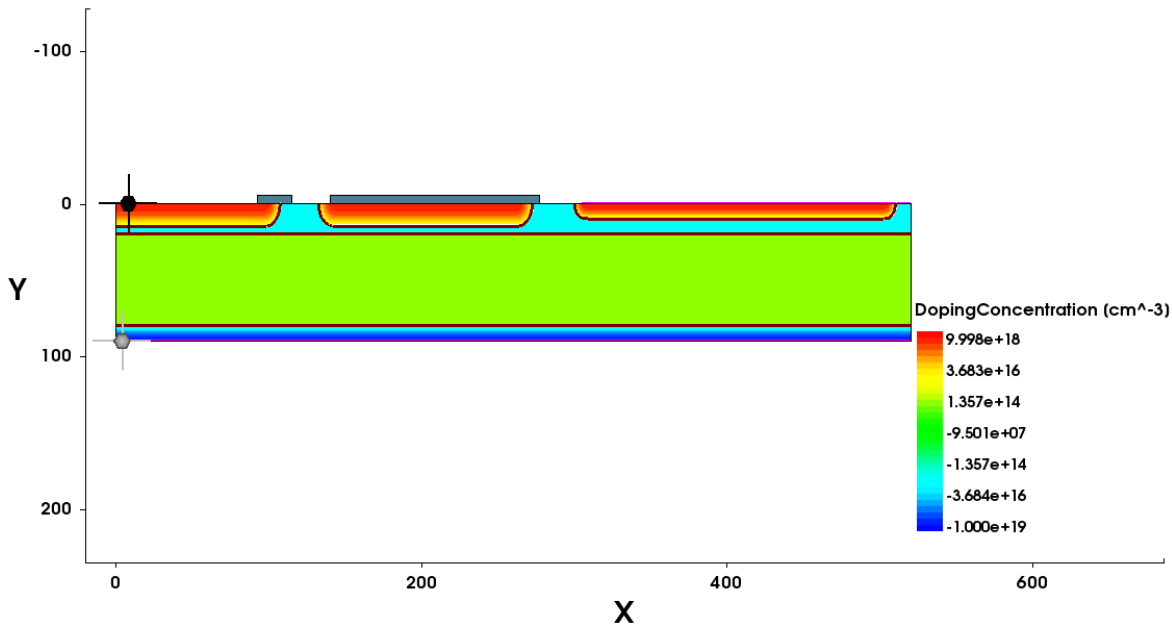


Figure 78 Maximum marker (black circle with cross hairs) and minimum marker (gray circle with cross hairs) shown to left of structure

You also can define constraints for the search pool in the Minimum/Maximum Field Value dialog box. To display this dialog box, choose **Tools > Min/Max Field Value**.

In the Minimum/Maximum Field Value dialog box, you can select certain regions or materials for the search, and you can define a 3D box limiting the search area.

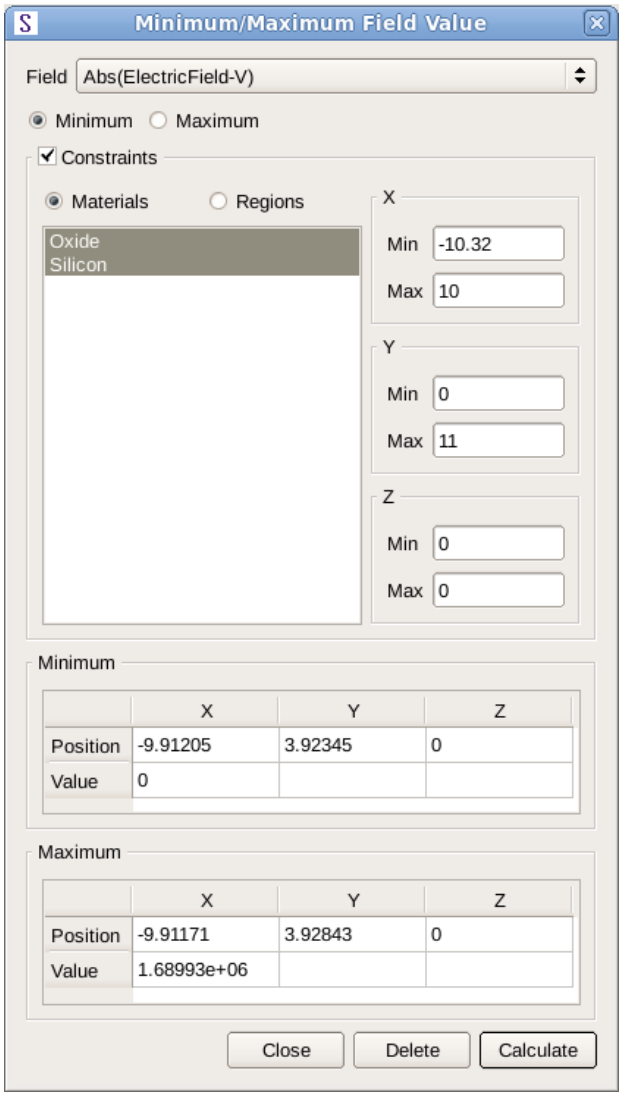


Figure 79 Minimum/Maximum Field Value dialog box

Changing Properties of Markers

You also can change the properties of the marker.

To change to the marker properties:

1. Choose **Edit** > **Preferences**.
2. In the User Preferences dialog box, expand **2D/3D** > **Fields** (see Figure 80).
3. Change the preferences as required.
4. Click **Save**.

You can change the color, the size, and the visualization of both the minimum and maximum markers. When you save the settings, they will be used in the next session of Sentaurus Visual.

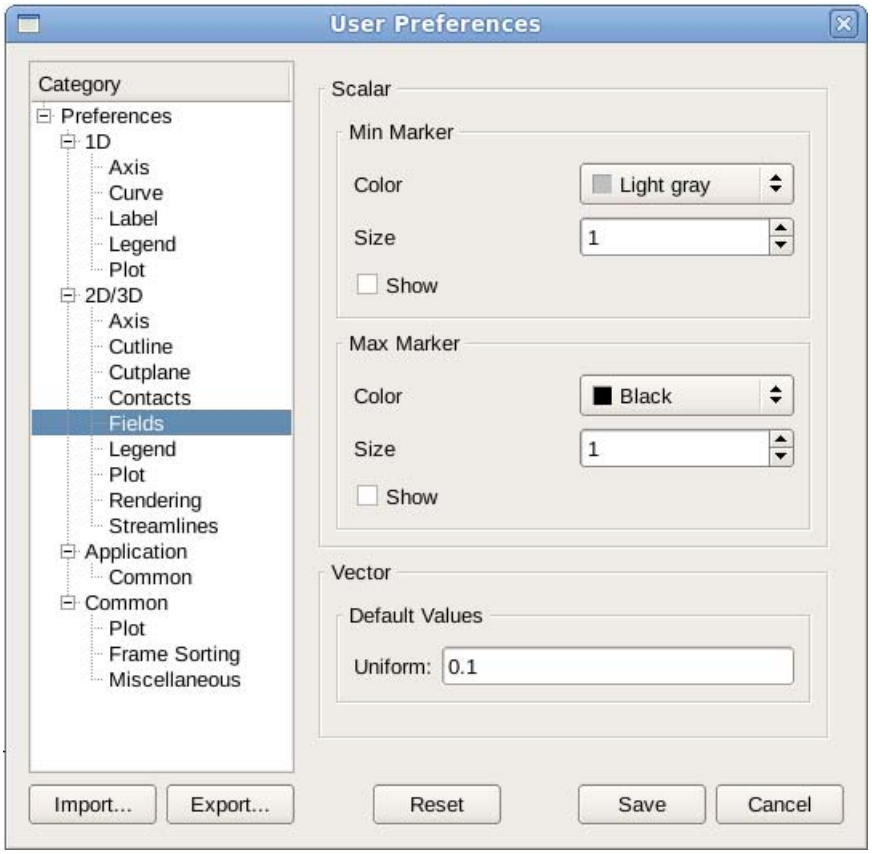


Figure 80 Available preferences for minimum and maximum markers

Value Blanking

Value blanking allows you to display only the required areas of interest in a plot. You can enter multiple constraints to blank out areas that meet the criteria.

To use value blanking, click the toolbar button. A dialog box is displayed (see Figure 81) where you can insert constraints on the required fields.

Value blanking keeps the enabled constraints even after closing the window. If you want to revert the changes, you must disable the specified constraint or reset all the constraints to return to the usual plot display.

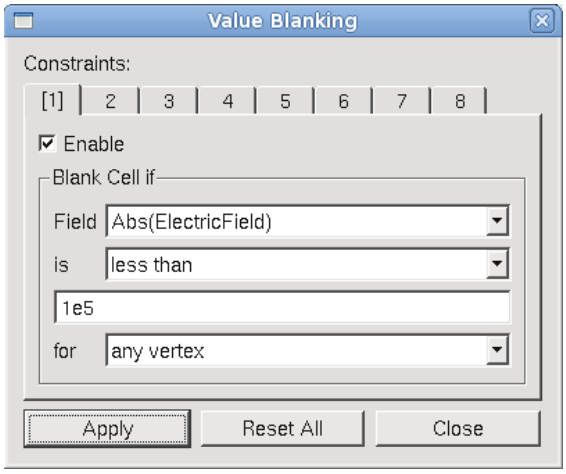


Figure 81 Value Blanking dialog box

Choosing Constraints

In the Value Blanking dialog box, you can create a maximum of 10 constraints including different field values. Each constraint creates a particular set of data to be blanked. The sets constructed can be united or intersected, depending of the option selected for the particular constraint.

For example, if cons1 defines set A and cons2 defines set B, the result set to be blanked C can be either $C = A \cup B$ or $C = A \cap B$ depending of the option selected in cons2 (that is, either the **Union** option or the **Intersection** option) (see Figure 82 on page 105).

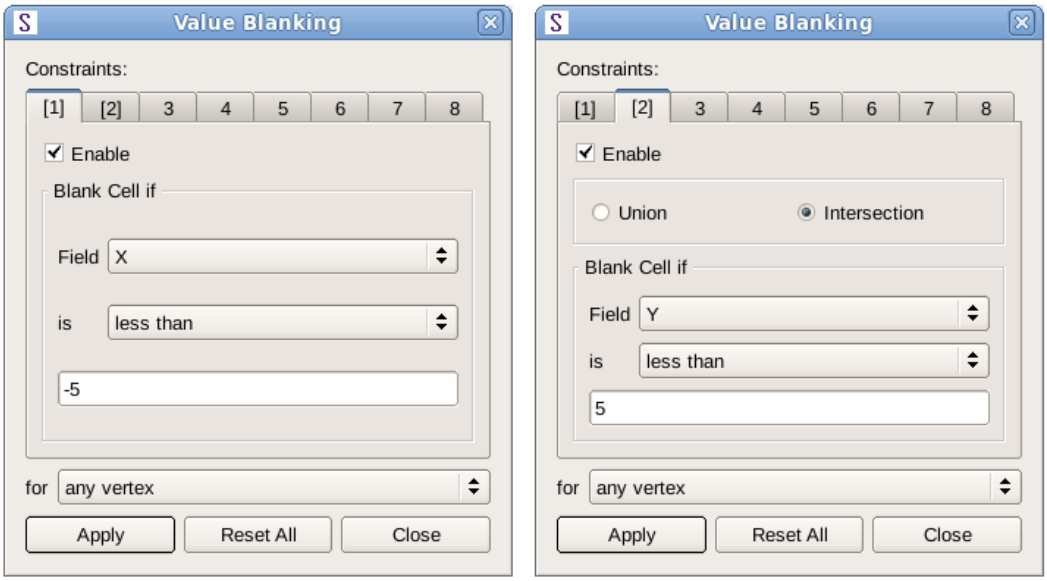


Figure 82 Value Blanking dialog box showing constraints

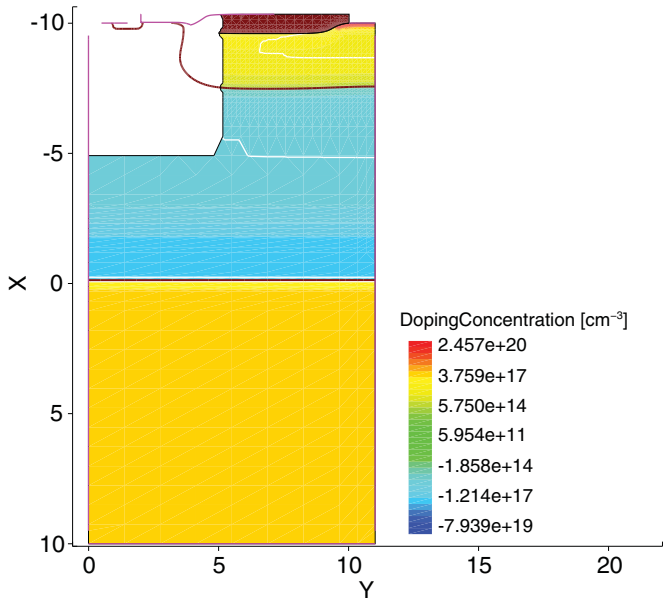


Figure 83 Blanked plot

Options for Value Blanking

The options for value blanking are available from the **for** list of the Value Blanking dialog box (see Figure 81 on page 104). The options are:

- **all vertices**
- **any vertex**
- **interpolate vertices**

Examples using these options are displayed in the following figures.

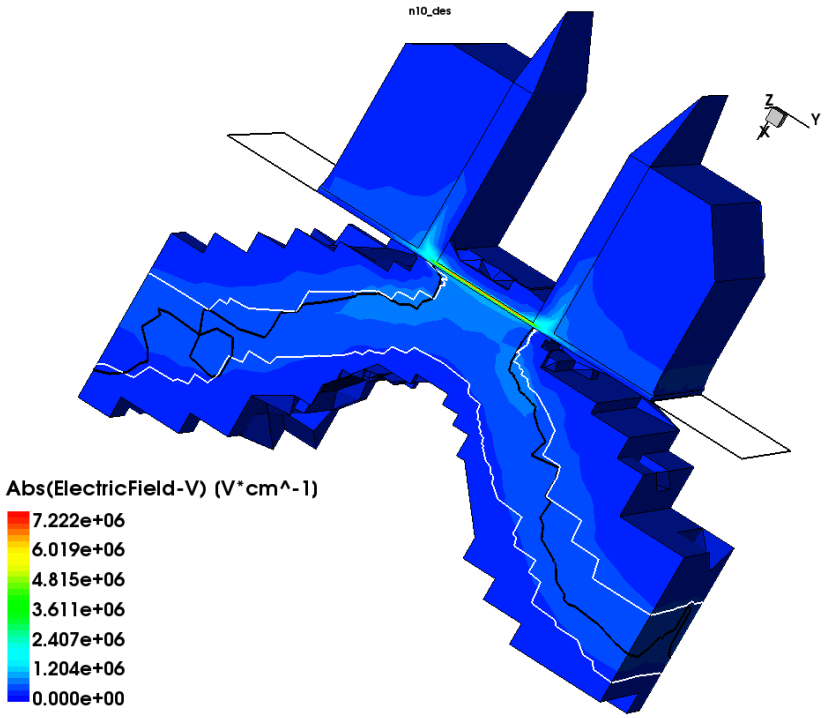


Figure 84 Example of value blanking using the all vertices option

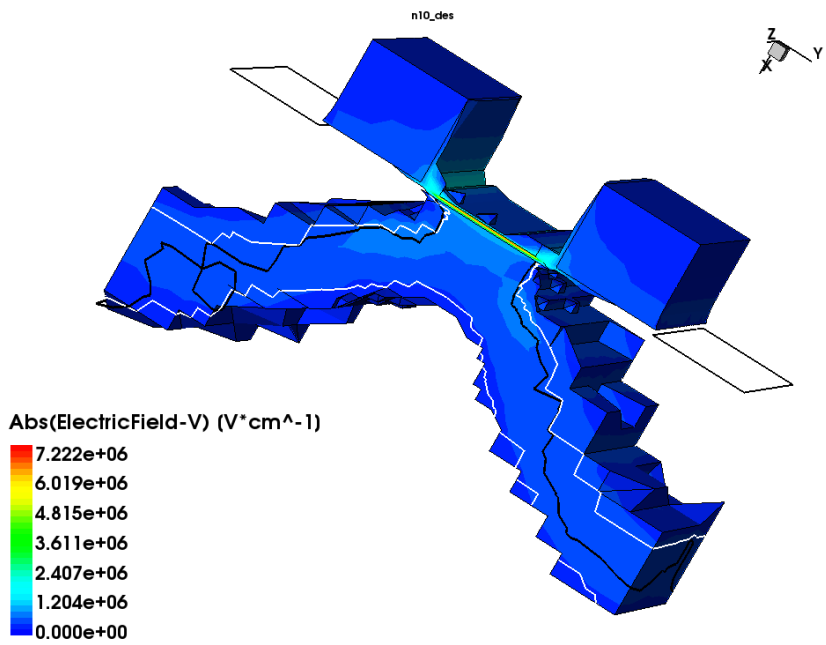


Figure 85 Example of value blanking using the any vertex option

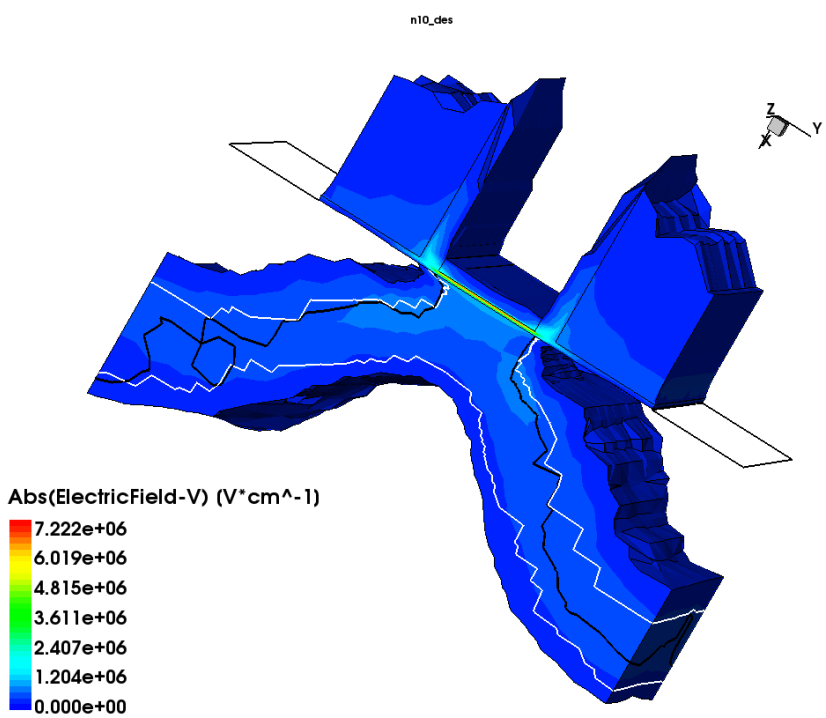


Figure 86 Example of value blanking using the interpolate vertices option

As shown in Figure 86 on page 107, the **interpolate vertices** option makes the surface of the field being blanked smoother than when the other options are selected.

Visualizing Deformation of Structures

Sentaurus Visual allows you to deform a structure according to a certain vector field. In general, the required vector field that will be used to deform the structure is Displacement and, in Sentaurus Visual, it is called Displacement-V (all vector fields have the suffix -V).

Every region that contains the selected vector field will be deformed. This means that every point that defines the region will be moved in the direction and the magnitude of the vector. The magnitude of the displacement can be modified by a factor of the vector. This value is 1.0 by default (no factor is applied).

You can deform structures using either the user interface or the `set_deformation` Tcl command (see set_deformation on page 270).

In addition, you can apply a deformation factor or create a new deformation plot in a group of plots using the `link_plots` command. However, if you want to apply a deformation operation but not to all members of a plot group, you must create a special linked group without specifying the deformation property (see link_plots on page 216).

To display the Deformation dialog box, choose **Tools** > **Deformation** (see Figure 87). The Deformation dialog box includes the following fields and buttons:

- The **Vector** field is used to select the vector field that will be used to deform the structure.
- The **Factor** field is used to specify the factor of the magnitude vector to be used.
- The **Reset** button reverts the structure to its original state.
- The **Apply** button applies the specified deformation to the current plot.
- The **Create Plot** button creates a new plot with the same structure already deformed.

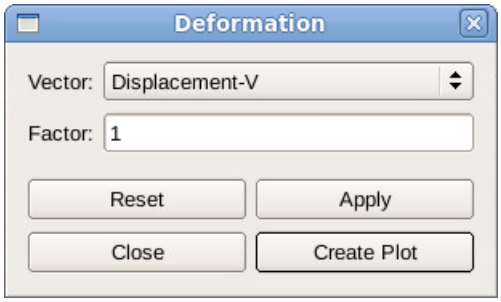


Figure 87 Deformation dialog box

By default, if the **Displacement-V** vector field is available, it will be selected automatically in the **Vector** field. If this vector field is not available, no vector field will be selected. By default, the factor value is 1.0.

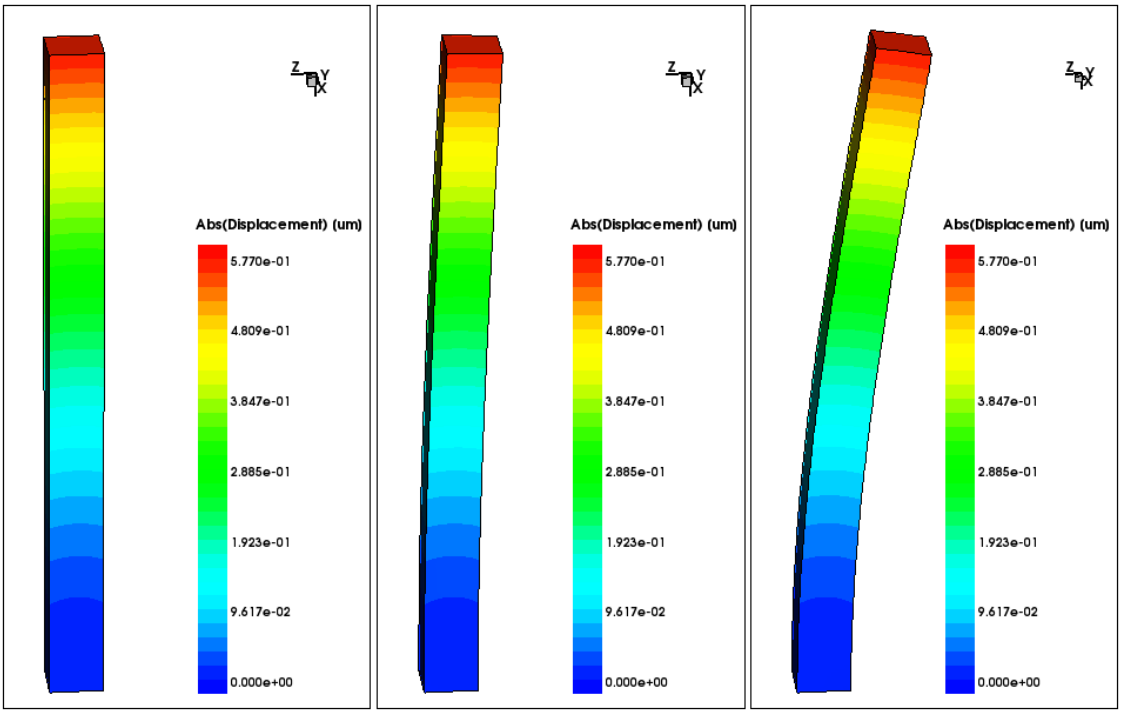


Figure 88 (*Left*) Original structure, (*middle*) structure deformed by a factor of 3, and (*right*) structure deformed by a factor of 10

Cutting Structures

Sentaurus Visual provides tools for generating xy and 2D cuts, custom cutlines, and cutlines or cutplanes orthogonal to an axis. Table 8 lists the available cutting tools.

Table 8 Tools for cutting structures

Toolbar button	Description
	Displays the Cutlines and Cutplanes dialog box, where you can generate non-orthogonal cutplanes or cutlines directly from a 3D plot, and you can cut in specific values.
	Creates a custom cutline in a 2D plot. The result is an xy plot of the selected field and datasets for all the fields on the cutline.

Table 8 Tools for cutting structures

Toolbar button	Description
X	Creates an orthogonal plot in one axis. The result is a 2D plot of the cutplane if cutting a 3D plot, or an xy plot from a cutline in a 2D plot. If an axis has a constant value, cuts for that axis are disabled.
Y	
Z	

NOTE In linked plots, the newly cut structures are created by frame order *not* load order.

Generating Precise Cutlines and Cutplanes

Advanced options such as cutting a 3D structure non-orthogonally by specifying a normal and an origin point, or creating a cutline directly from a 3D structure can be performed using the Cutlines and Cutplanes dialog box (see Figure 89 on page 111). It allows a greater degree of precision than using mouse operations to generate cuts that require an exact point in the structure.

To display the Cutlines and Cutplanes dialog box, click the toolbar button.

In addition for 2D plots, other cuts can be performed such as a polyline cut or a cut along boundaries. The Cutlines and Cutplanes dialog box provides various **Add by Click** buttons that allow you to click a point in the plot and to add it to a specific text box. The point clicked in this way will be marked in the plot.

For 2D and 3D structures, when **Cut Type** is set to **Cutline**, you can specify the regions to use as the source for the cutlines and specify the resulting target plot where the cutline curve will be displayed.

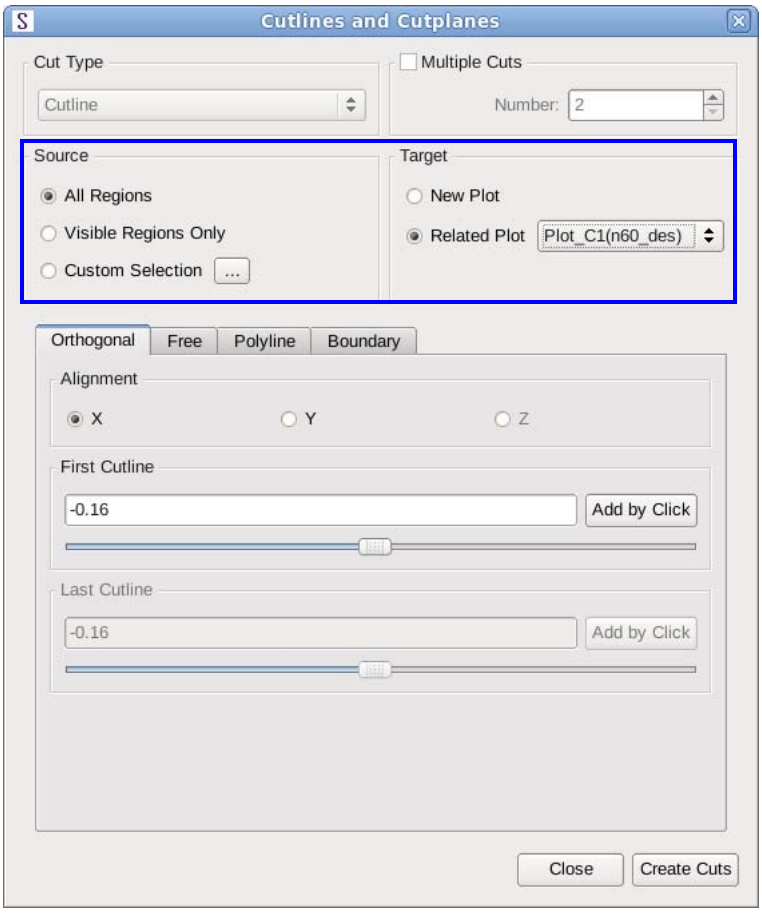


Figure 89 Cutlines and Cutplanes dialog box

For the source, the following options are available under Source:

- **All Regions** (default): All regions are cut.
- **Visible Regions Only**: All visible regions are cut.
- **Custom Selection**: You select the regions or materials to be cut in the Custom Selection dialog box (see Figure 90 on page 112).

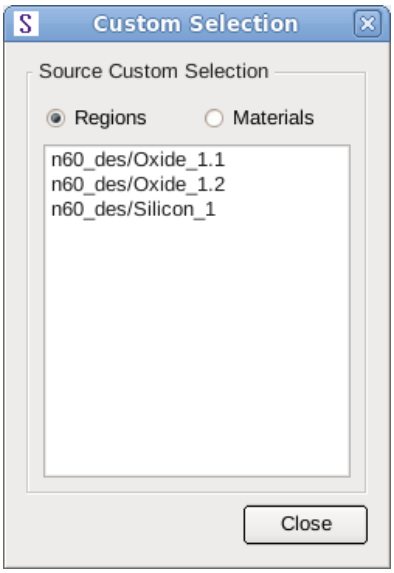


Figure 90 Custom Selection dialog box

The default target plot is **Related Plot** (see Figure 89 on page 111). A *related plot* refers to any xy plot that has already displayed at least one cutline dataset with the same type of the current cut. For example, if you perform an orthogonal x-cut, a related plot would be a plot created to display another orthogonal x-cut.

The list of the **Related Plot** option shows the plots associated with the cutline type specified:

- For 2D plots, this is for all cutline types (X, Y, Z, or Free), including polyline cuts and cuts along boundaries.
- For 3D plots, this is for free-type cutlines only.

You can use the User Preferences dialog box (see Figure 91 on page 113) to override the default option of the Cutlines and Cutplanes dialog box.

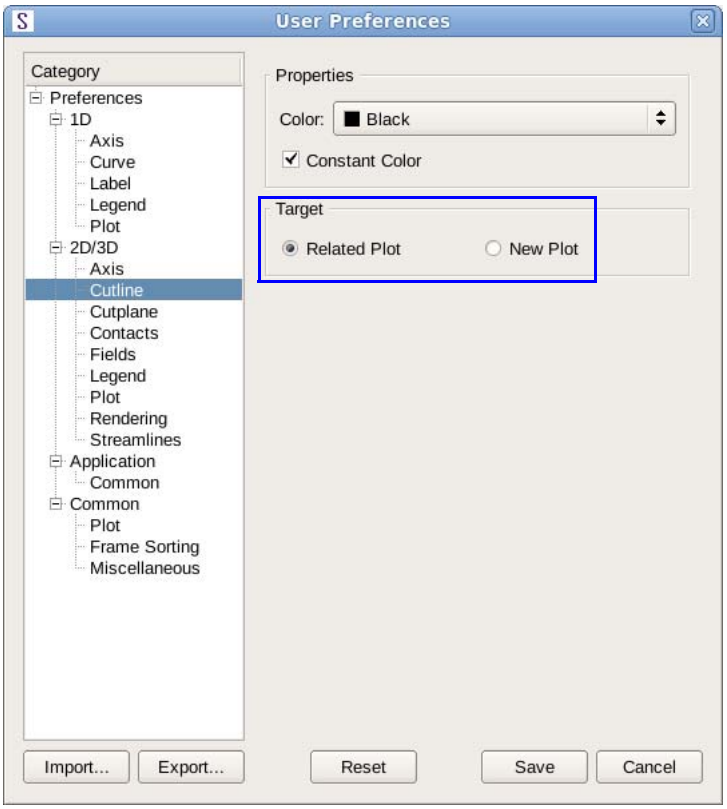


Figure 91 Specifying the default for cutline target plots in the User Preferences dialog box

Cutlines in 2D Plots

Creating a new cutline is as easy as selecting an axis and a point in the plane to perform an orthogonal cut or drawing a line using the custom cutline button. The result is a new xy plot as shown in Figure 92 on page 114.

In the generated xy plot, the y-axis will be displayed in logarithmic scale if the active scalar field in the original 2D plot is visualized using one of the following scales: logarithmic (Log), logarithmic of the absolute (LogAbs), or hyperbolic arcsine (Asinh). Otherwise, the y-axis will be displayed in linear scale.

If the active scalar field uses a custom scale, the y-axis will also be displayed in linear scale. See Visualizing Fields on page 60.

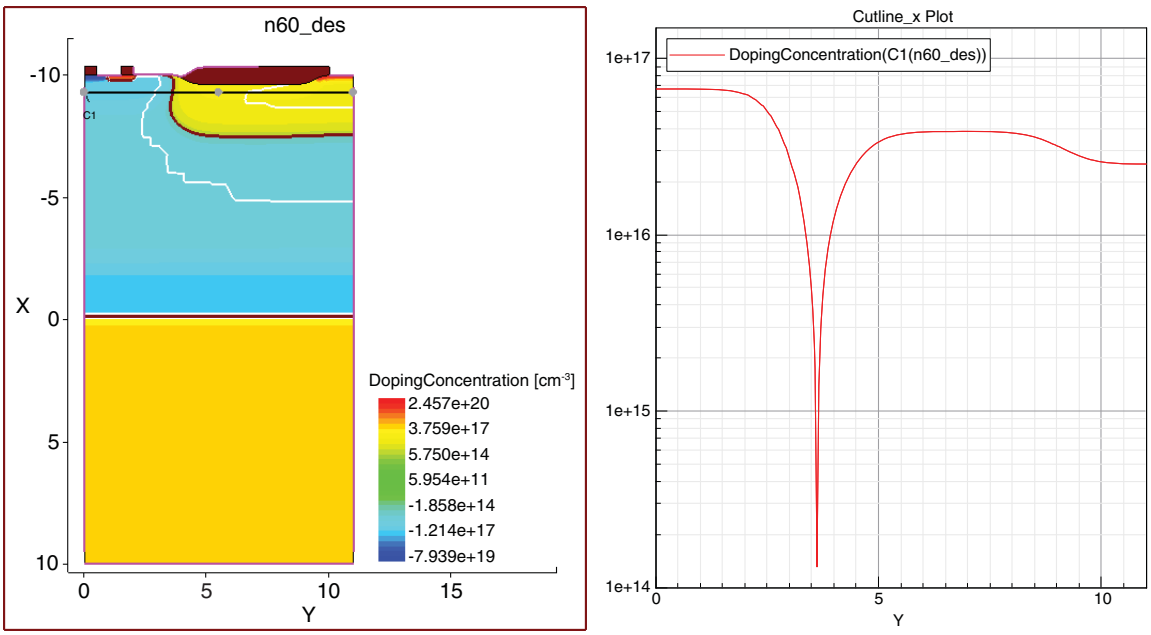


Figure 92 (*Left*) Cutline drawn on 2D plot and (*right*) xy plot generated from cut

Manipulating Cutlines

Cutlines can be moved and resized by dragging the cutline handles (the circles at the ends of the cutline), and the cutline plot is updated automatically.

To delete a cutline, select the cutline and press the Delete key.

NOTE The xy plot created is not deleted. Deleting the xy plot does not delete the cutline in the 2D plot.

Polyline Cuts in 2D Plots

A polyline cut is the union of two or more cutlines where the end point of one is the start point of the other. Polyline cuts can be created selecting the **Polyline** tab in the Cutlines and Cutplanes dialog box (see Figure 93 on page 115).

The points that define the polyline can be added by using the keyboard, or using the fields in the Point group box, or clicking directly in the plot after clicking the **Start Add by Click** button. When a point is added, the position of the point is marked on the plot. For example, the points added in Figure 93 will produce the marks shown in Figure 94 on page 116.

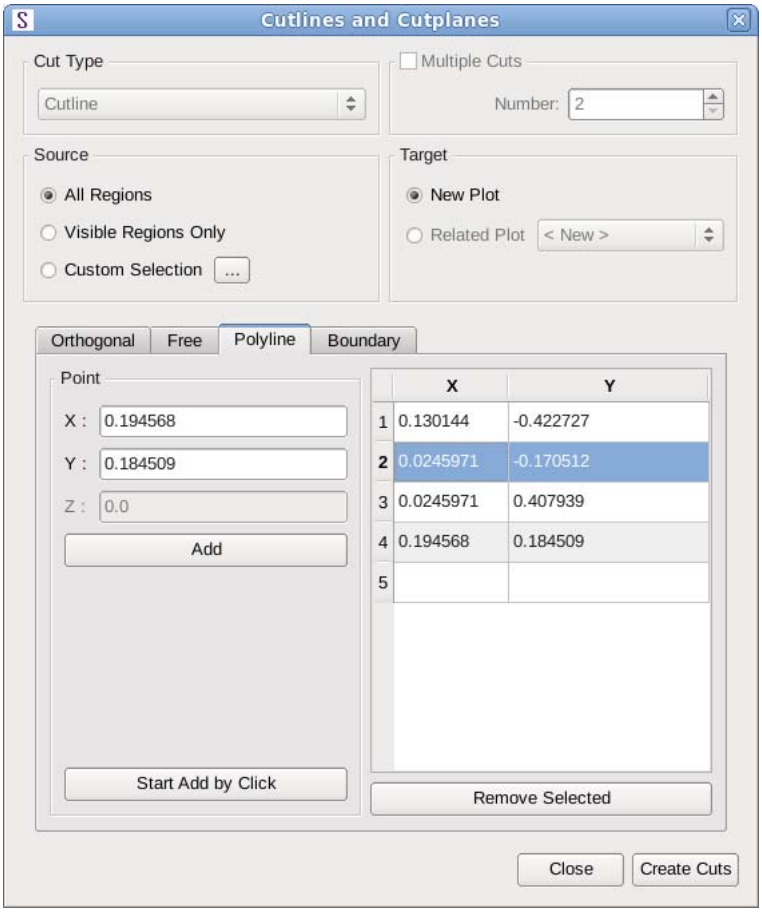


Figure 93 Polyline tab

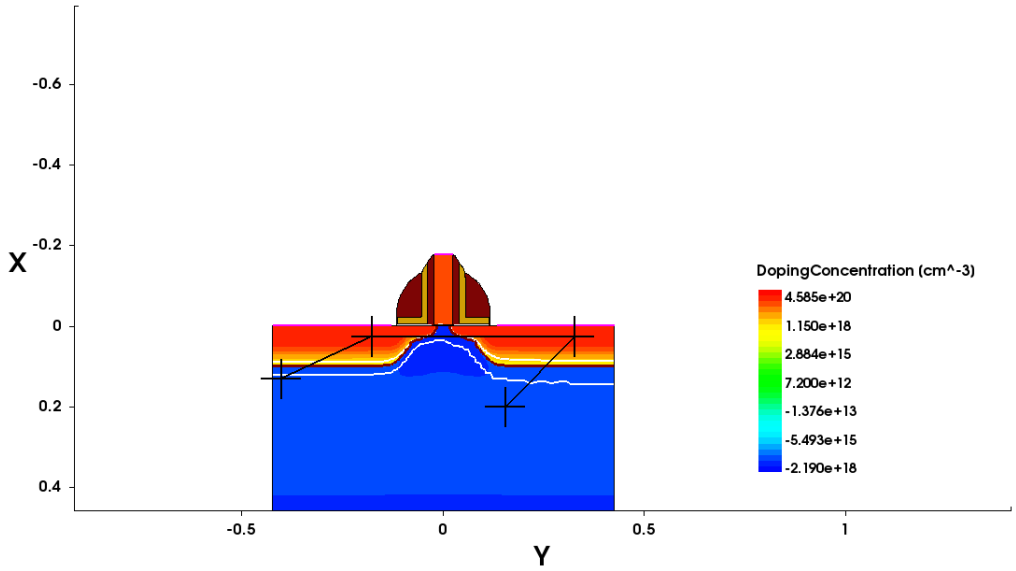


Figure 94 Marks indicate position of points generated from values in Figure 93

When all the points have been added, the cut is created by clicking the **Create Cuts** button. An xy plot is created immediately, showing the active field versus distance of the line. In addition, a new dataset is created containing the values along the line in all fields. Figure 95 shows the plot created.

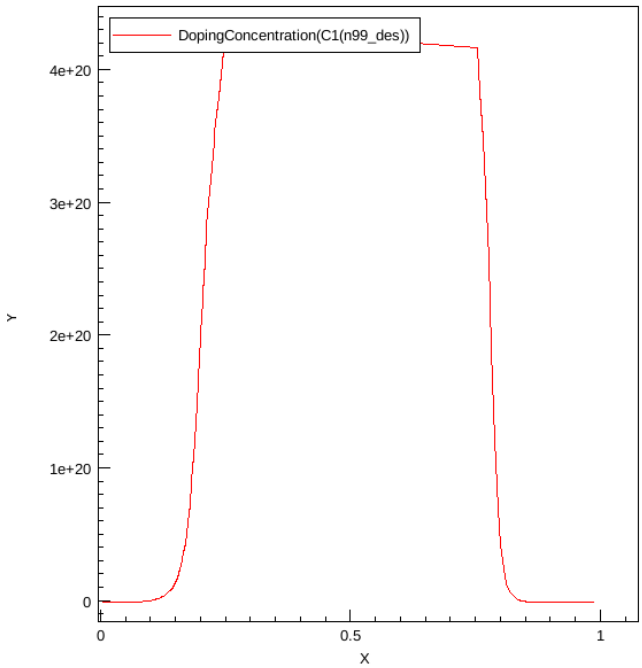


Figure 95 New xy plot created from polyline cut

Manipulating the Polyline

The location of each point can be changed by dragging its handle to a new position. This will update the values of the already created xy plot using the new positions of the points. Other properties such as the line style, color, and size can be changed on the **CutPolyline Properties** tab of the Properties panel, when the polyline is selected.

Cutting Along Boundaries in 2D Plots

You can cut along boundaries in 2D plots using a simple wizard, which is on the **Boundary** tab of the Cutlines and Cutplanes dialog box. Alternatively, you can use the `create_cut_boundary` command (see create_cut_boundary on page 153).

Step 1: Selecting Regions or Materials

Select the target regions or materials of interest. If you do not know which regions or materials you need, you can choose all of them and define them later.

To select regions or materials:

1. Select either the **Regions** or **Materials** option.
2. Move the available regions or materials to the Selected pane as required.
3. When you are finished, click **Next**.

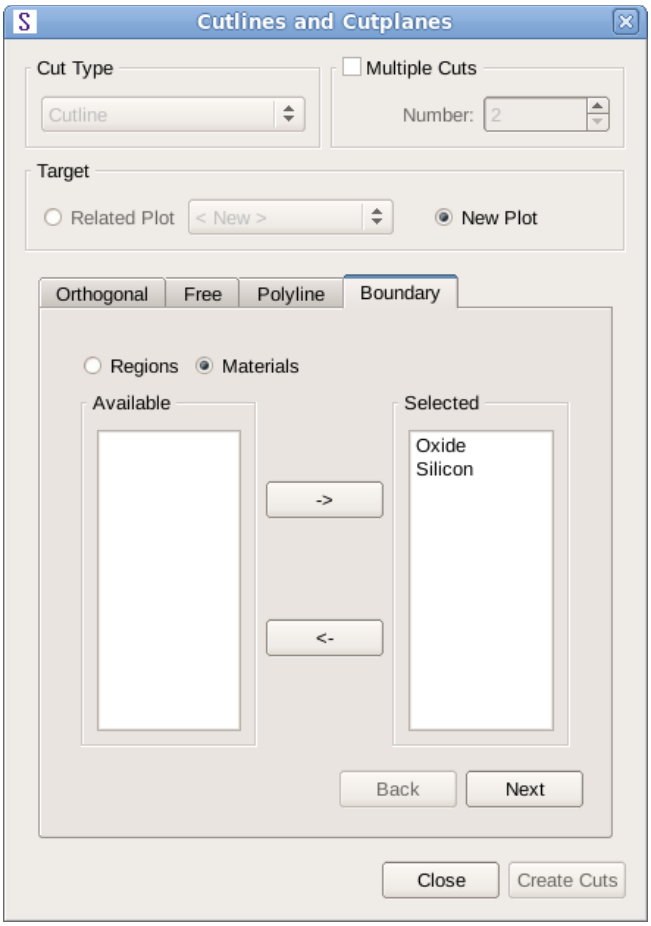


Figure 96 Step 1: selecting regions or materials

Step 2: Adding Vertex Points

Add the vertex points through which the line will pass. The first point and last point added will be the start and end of the line along the boundaries. The line will also pass through any middle point added in the respective order. If more than one path is possible, Sentaurus Visual will choose the shortest path along the available regions or materials selected in the past step.

To add points:

1. Add points in one of the following ways:

   a) Click the **Start Add by Click** button (which changes to the **End Add by Click** button). Click to add points inside the 2D plot. When you have finished adding points, click the **End Add by Click** button.

b) Use the Add Point fields to enter the x- and y-coordinates for a point. Click the **Add Point** button. The point is listed in the Selected Points pane. Continue to add points as required (see Figure 97).

In both cases, if the point added is not on a boundary, Sentaurus Visual selects the nearest boundary to that point.

2. When you have finished adding points, click **Next**.

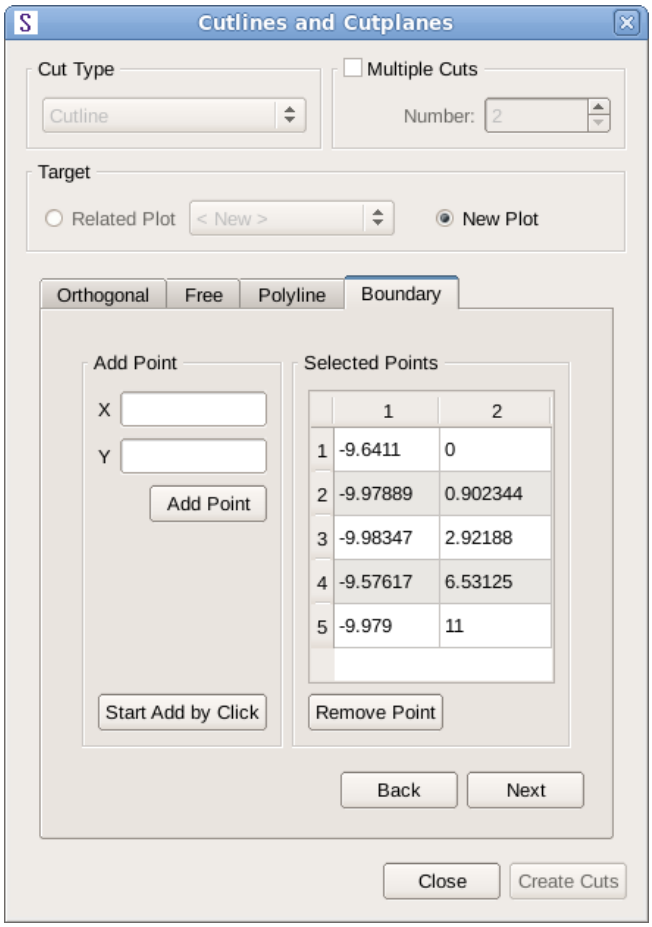


Figure 97 Step 2: adding vertex points

Step 3: Choosing Segment Regions

After you have added the vertex points, Sentaurus Visual divides the resulting line into various segments defined by the intersections of neighboring regions. In each segment, the required region from which the data will be extracted can be chosen by clicking the cell in the Region column (column 3), as shown in Figure 98 on page 120.

When all the regions in each section are selected, the cut can be created by clicking the **Create Cuts** button. The default segment regions are chosen by the order of regions or materials previously set.

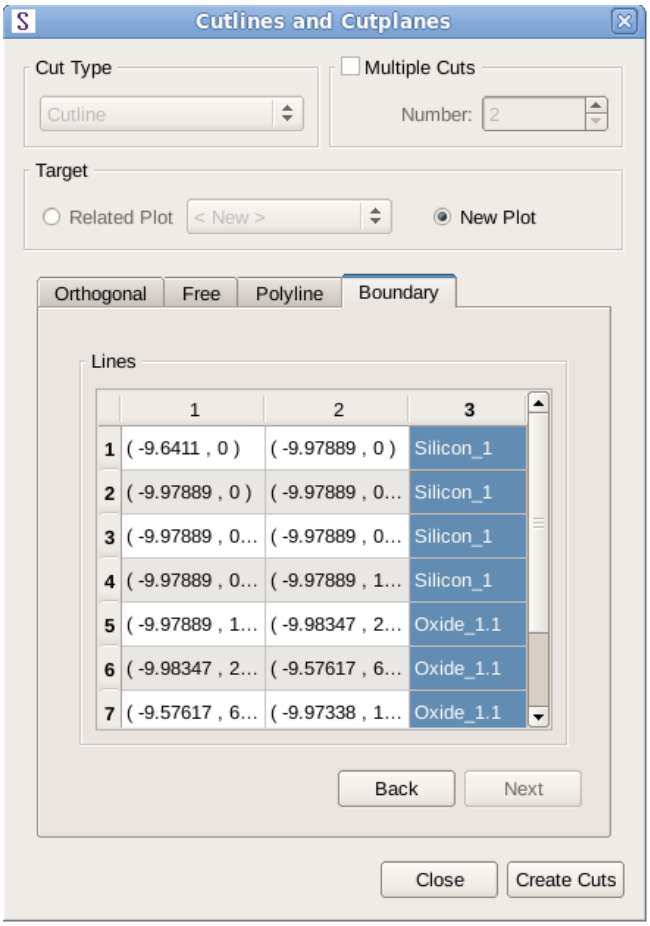


Figure 98 Step 3: choosing segment regions

The resulting plot shows the values along the selected regions of the active field versus distance. In addition, a dataset containing all the respective fields is created (see Figure 99).

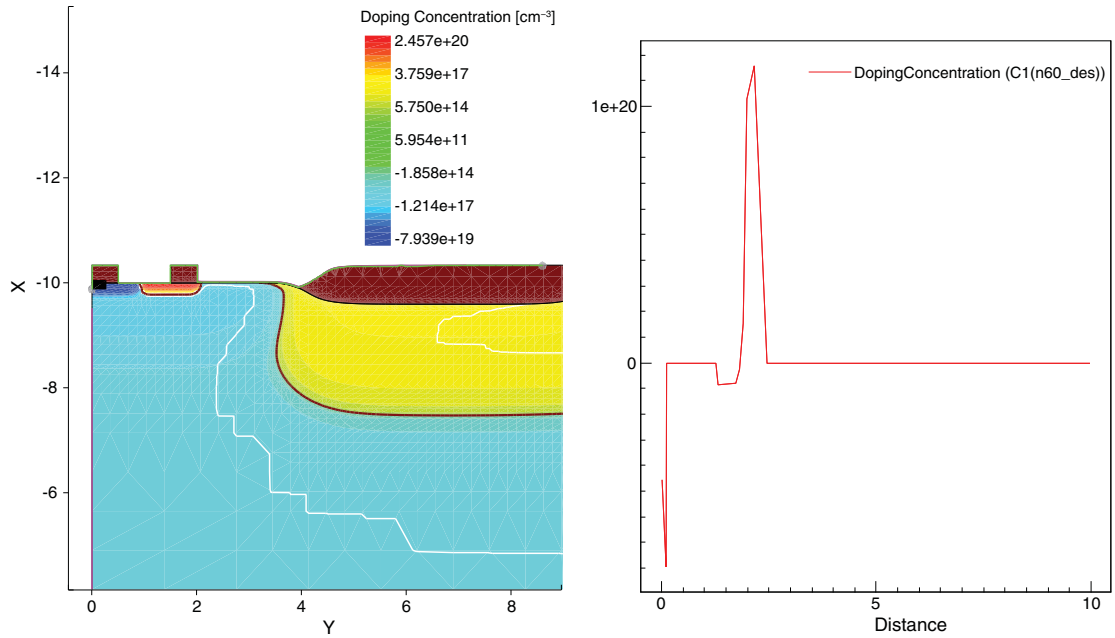


Figure 99 (*Left*) Original 2D plot and (*right*) resulting xy plot from cutting along boundaries

Changing Properties of Cutline Along Boundaries

Some properties such as the color, size, and type of the line can be changed in the **Cutline Properties** tab, which is displayed when the line is selected in the plot. The color and visibility of the handles of the first and last points can be changed as well.

2D Projection Plot

You can create 2D projection plots based on 3D plots. The resulting plot is either the maximum or minimum field value projected to one plane aligned to the orthogonal axes.

You can create a 2D projection of a 3D plot by either choosing **Tools** > **Create Projection**, which displays the 2D Projection dialog box (see Figure 100 on page 122), or using the `create_projection` command (see create_projection on page 162).

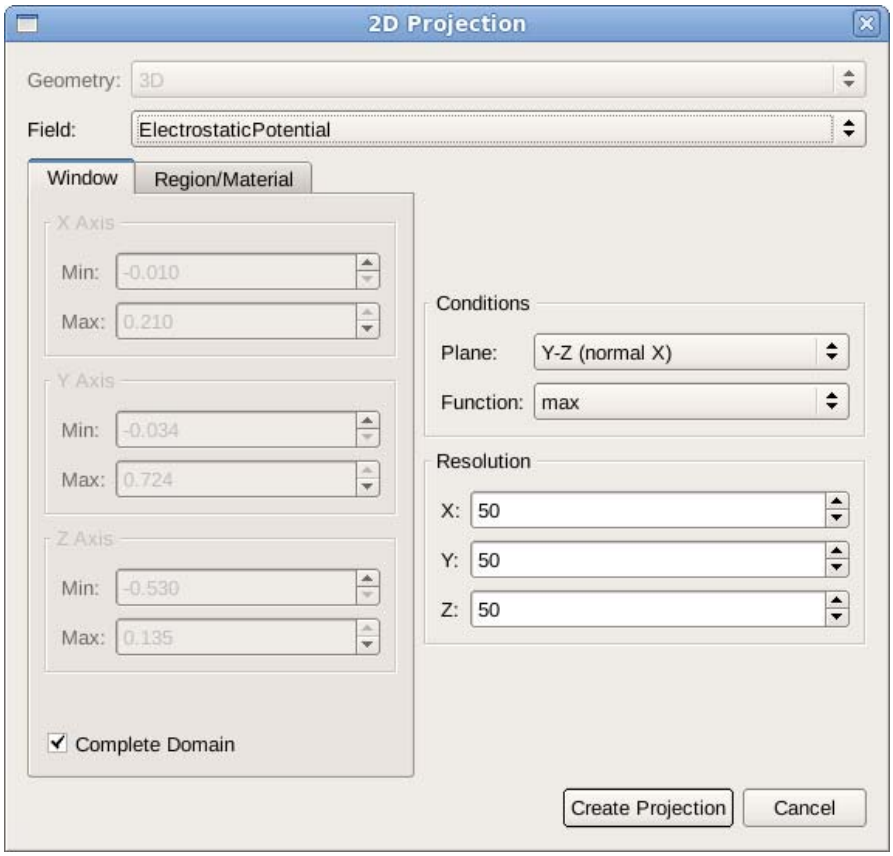


Figure 100 2D Projection dialog box

In the Conditions group box, you can choose the plane for the projection and the function to apply (either maximum or minimum). The Resolution variables define the number of points to consider in each axis. More resolution means a more exact plot, but it will take longer to process the data. The projection can be performed in all domains, or in a smaller window defined on the **Window** tab, and can contain all regions and materials, or only specific ones.

When the variables have been defined, the projection can be created by clicking the **Create Projection** button.

Figure 101 shows the final plot for the maximum value of a zy projection.

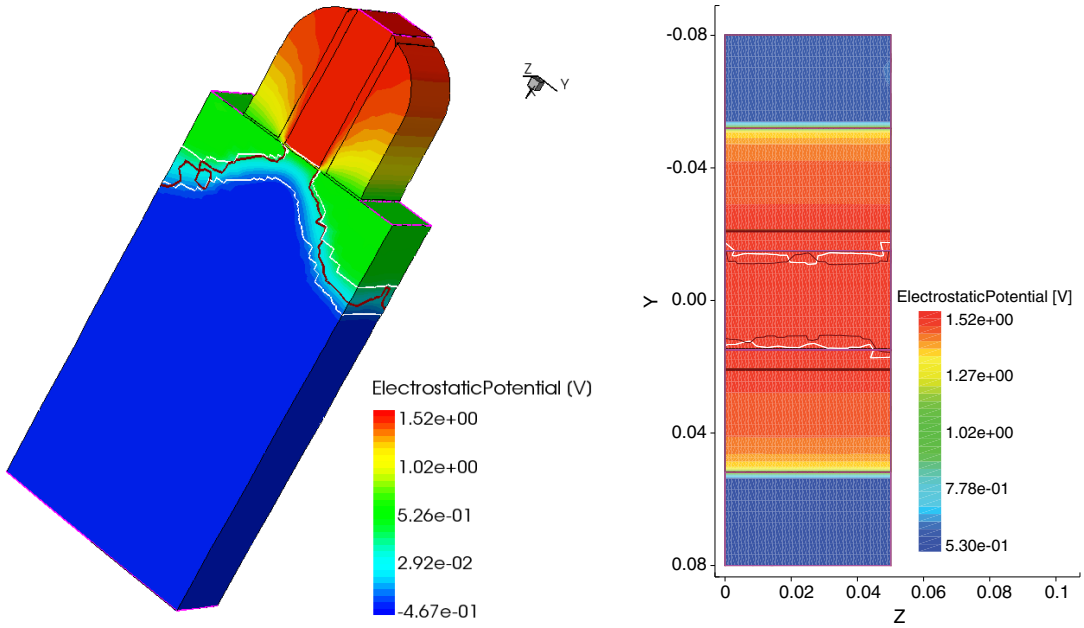


Figure 101 (*Left*) Original 3D plot and (*right*) 2D projection of the 3D plot

Cutplanes in 3D Plots

In 3D plots, orthogonal cutplanes can be created by selecting a cut axis and then clicking the required point of the plot. The result is a new 2D plot with the same fields as the original plot as seen in Figure 102 on page 124. Such a 2D plot can be cut further by a cutline to generate an xy cut.

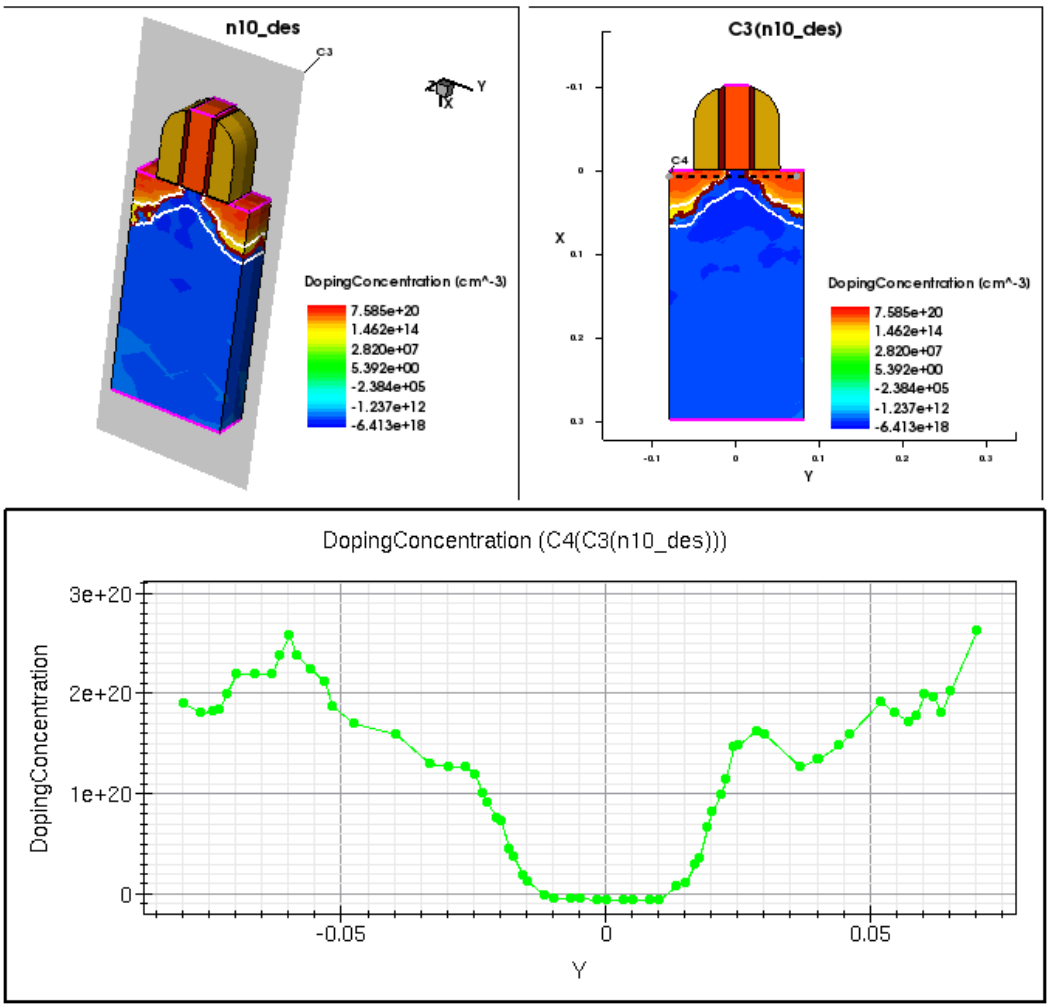


Figure 102 Cutplane in a 3D plot and the generated 2D plot, which is cut further to generate an xy plot

Cutplanes also can be moved by dragging, and the 2D plot is updated automatically. In addition, xy plots created from such 2D cuts are updated automatically.

To delete a cutplane, select the cutplane and press the Delete key.

NOTE Deleting the 2D plot does not delete the cutplane in the 3D plot.

NOTE The mesh shown on the cutplane is recalculated by triangulating the resulting points of the cut, which means, for example, that an axis-aligned cut of a rectangular mesh shows a triangular mesh.

Extracting the Path of Minimum or Maximum Values of a Scalar Field

NOTE This operation applies only to 2D plots.

Sentaurus Visual can extract the path of either the minimum or maximum values of a specified scalar field along the horizontal axis. This extraction does not refer to a specific axis, so interchanging the x-axis and y-axis of a 2D plot will lead to different results.

You can use a Tcl command to extract the path (see extract_path on page 180) or the corresponding dialog box, which is available from **Tools** > **Extract Path** (see Figure 103).

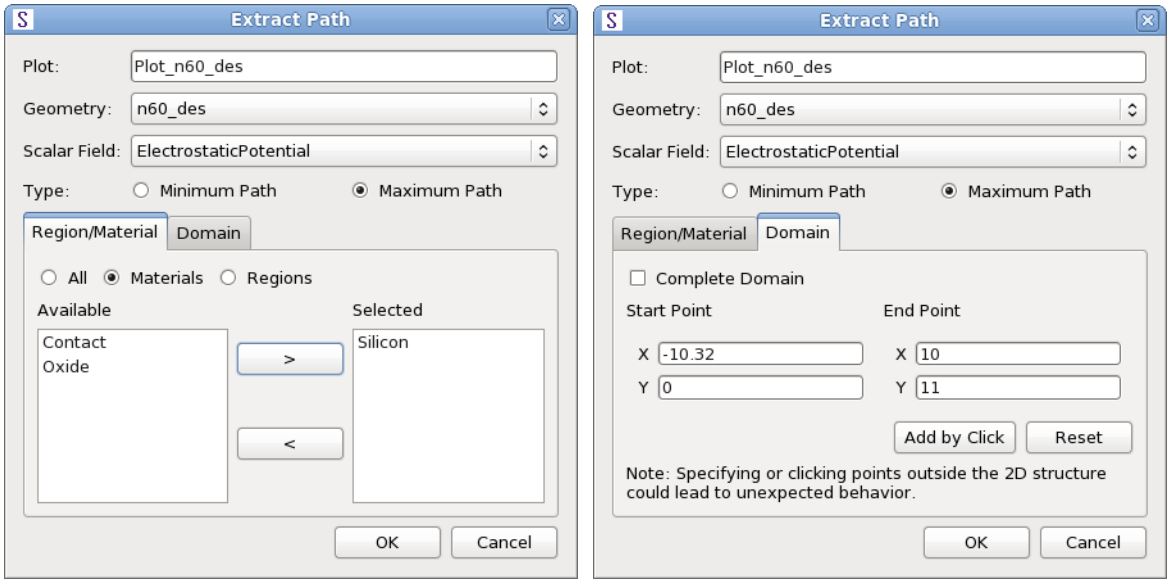


Figure 103 Extract Path dialog box showing (*left*) Region/Material tab and (*right*) Domain tab

To extract a path:

1. Leave the plot name in the **Plot** field. This is the name of the active plot.
2. Select the geometry from the **Geometry** list.
3. Select the scalar field for the extraction from the **Scalar Field** list.
4. Choose which values you want to extract: **Minimum Path** or **Maximum Path**.
5. On the **Region/Material** tab, you select either to extract the path over all materials and regions (the **All** option), or selected materials, or selected regions.

   The materials and regions shown in the Available pane depend on which ones are present in the 2D plot.

6. Optionally, if you require greater precision, click the **Domain** tab to specify the start and end points of a smaller window of analysis (see Figure 103 on page 125 (*right*)).

7. Click **OK**.

The algorithm for the extraction involves recalculating the mesh for a 2D structure, normalizing the mesh to the smallest cell width in the horizontal direction. However, if this recalculation exceeds millions of divisions, Sentaurus Visual resolves this to one million divisions to maintain tool performance (see Figure 104).

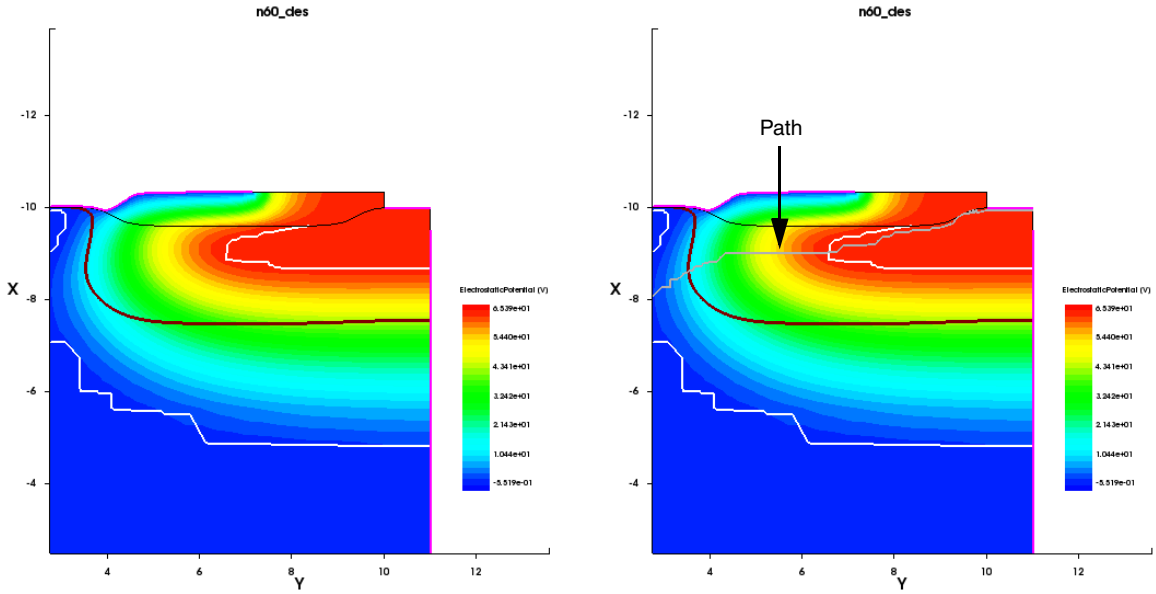


Figure 104 (*Left*) Original 2D structure and (*right*) extracted path over the entire 2D structure

The extraction results in the creation of a new geometry in the 2D structure that behaves like an interface region, allowing you to visualize the field values in the path, even if the main geometry does not display field data (see Figure 105 on page 127).

See Visualizing Fields on page 60 for more information about visualizing scalar fields.

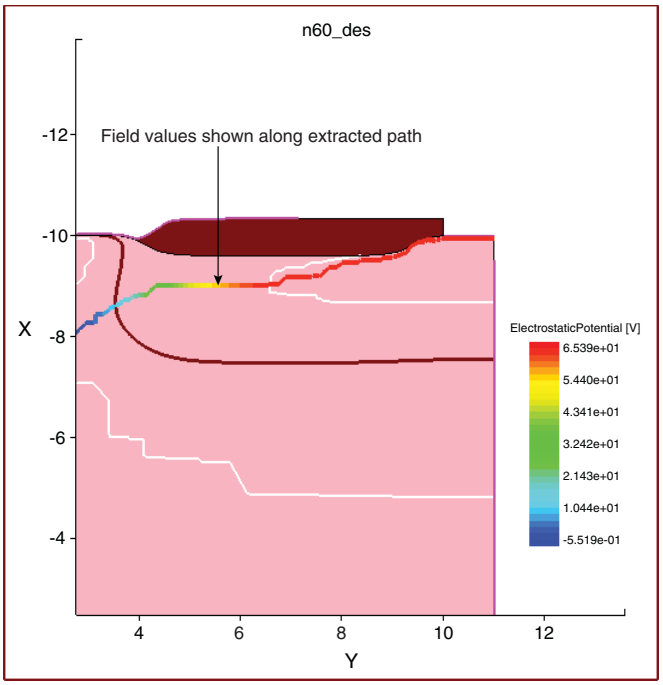


Figure 105 Field values along the extracted path, with main geometry displaying no field data

When you extract a path using the Extract Path dialog box, which is not a *cutting* operation, in addition to the path shown in the 2D plot, a regular xy plot is generated showing a curve of the extracted path field (see Figure 106). This xy plot is generated using the `create_curve` and `create_plot` commands.

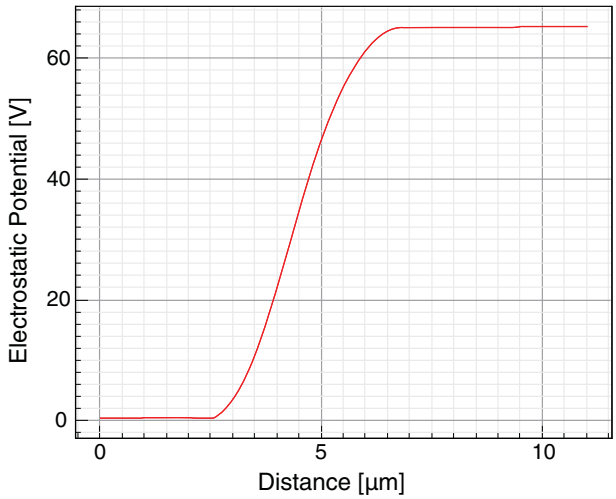


Figure 106 Extracted path displayed as an xy plot; curve represents the banded field (ElectrostaticPotential)

Surface Plots

A surface plot is a 3D plot generated from a 2D dataset (or plot), where the constant component (the z-axis if the original structure plane is the xy plane) is filled with an existing scalar field. As the new dataset contains the three components (x-axis, y-axis, and z-axis) defined, it behaves as a typical 3D dataset. If a 3D dataset is created, it will be shown as a new plot.

The new 3D dataset will contain the same regions and fields as the source dataset, which can be independent of the source plot. In addition, Sentaurus Visual recalculates the junction line and the depletion region, to show them according to the new surface.

The created 3D surface plot inherits the current field with filled contour bands and the visibility options of all regions. This means that the surface plot hides regions that are not shown in the source plot.

Creating Surface Plots

You can create a surface plot using either the Surface Plot dialog box (select **Tools** > **Surface Plot**) or the `create_surface` command (see create_surface on page 166).

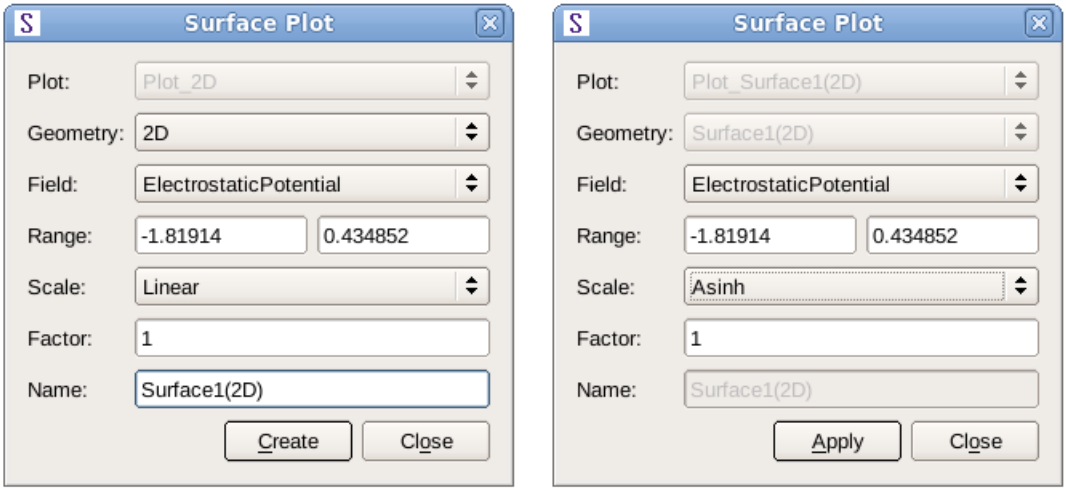


Figure 107 Surface Plot dialog box: (*left*) creating a surface plot and (*right*) modifying the newly created surface plot

In the Surface Plot dialog box, you can choose the geometry of the plot (if there is more than one), the field to be used, the range, the scaling type, the factor to define how the values of the field will affect the constant component, and the name of the new dataset generated.

After the new surface plot is created, the dialog box does not close. It changes appearance to allow you to modify the last generated surface plot (see Figure 107 on page 128, *right*). Some fields are disabled. The remaining fields can be changed to fine-tune the surface plot. With each change that is applied, the plot is updated.

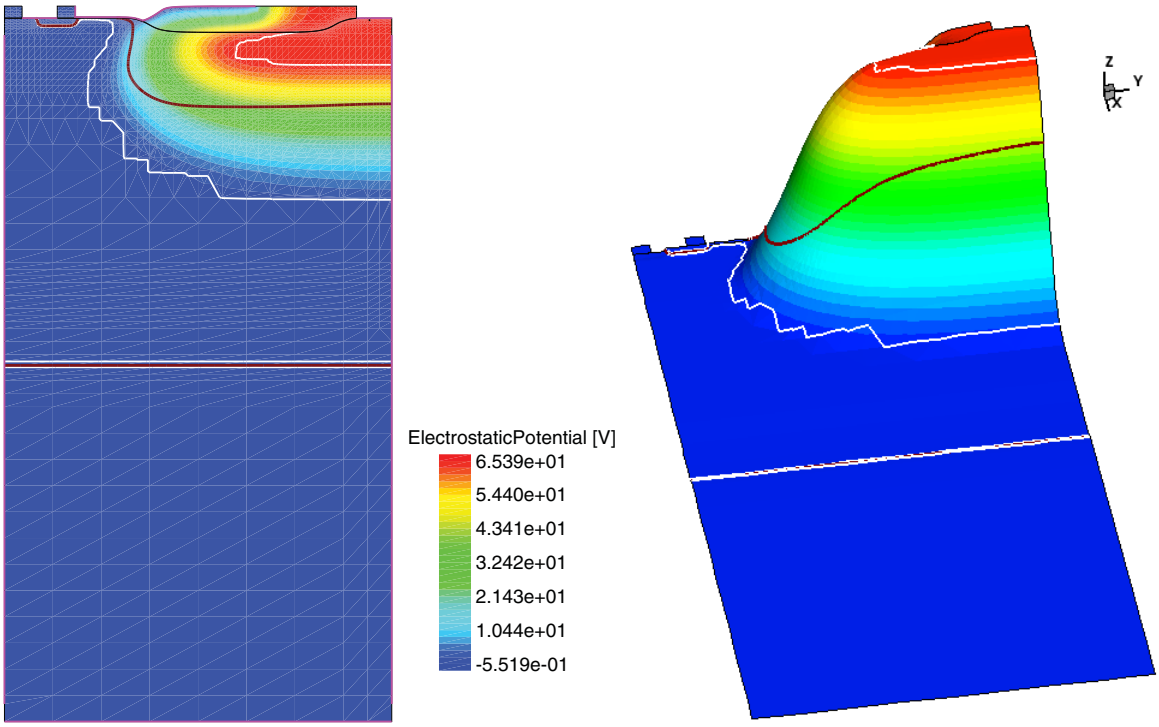


Figure 108 (*Left*) Two-dimensional source plot and (*right*) generated surface plot using the ElectrostaticPotential field

Isosurfaces and Isolines

Sentaurus Visual can extract isosurfaces and isolines from 3D and 2D structures, respectively. These iso-geometries are extracted using a constant value (isovalue) over a specified field and structure. The extracted iso-geometry is displayed in the same plot as the source geometry, such as an overlay plot. The new iso-geometry is divided into different regions and contains the same fields as the source geometry. This means that it is possible to display the same or different fields in both geometries. By default, the new iso-geometry has a constant color to help identify it easily. A plot can contain as many iso-geometries as you want.

The new iso-geometry does not contain any line or particle region from the source geometry and contains only the regions that have values of the specified field.

Creating Iso-Geometries

You can create a new iso-geometry using either the Create Isovalue Geometry dialog box (**Tools** > **Create Isovalue**) or the `create_iso` command (see create_iso on page 160).

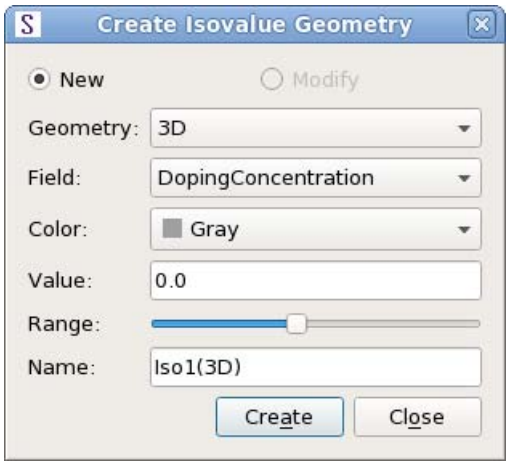


Figure 109 Create Isovalue Geometry dialog box

To create a new iso-geometry using the Create Isovalue Geometry dialog box (the **New** option is selected by default):

1. Choose the geometry of the plot (if there is more than one).
2. Choose the field to be used.
3. Choose the color with which to display the new iso-geometry.
4. Type the isovalue to use to build the iso-geometry.
5. Use the **Range** slider to identify where the value lies in the range of the field.

   This can be used by the selected interpolation to display the field.
6. Type the name of the new iso-geometry (or dataset).
7. Click **Create**.

NOTE After you create an iso-geometry, the Create Isovalue Geometry dialog box continues to be displayed and changes automatically to the modification mode, so that you can modify the iso-geometry if required (see Modifying Iso-Geometries on page 132).

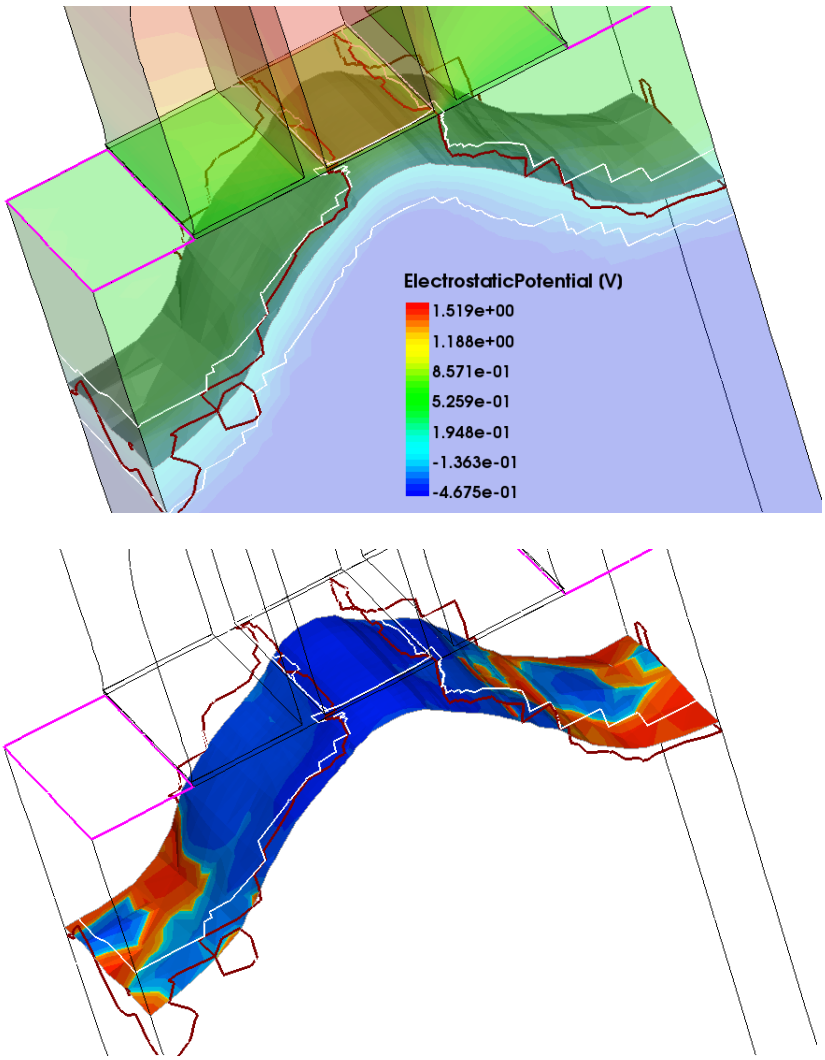


Figure 110 (*Top*) Source geometry with translucency showing the new iso-geometry generated (ElectrostaticPotential = 0 V) and (*bottom*) iso-geometry displaying the DopingConcentration field

Modifying Iso-Geometries

You can modify existing iso-geometries using either the Create Isovalue Geometry dialog box (**Tools** > **Create Isovalue**) or the `create_iso` command (see create_iso on page 160).

To modify an iso-geometry using the Create Isovalue Geometry dialog box (see Figure 111):

1. Ensure the **Modify** option is selected to enable the modification mode.
2. Change the fields as required.

   NOTE You can change all the fields except the name of the iso-geometry.

3. Click **Apply**.

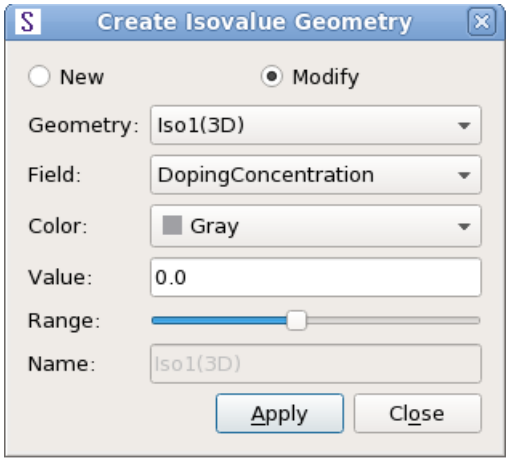


Figure 111 Create Isovalue Geometry dialog box in modification mode

Streamlines

Streamlines are a family of curves that are instantaneously tangent to the velocity vector of the flow. Sentaurus Visual allows you to visualize these streamlines for the available vector fields in 2D or 3D plots.

Streamlines are created only in the active plot by default, even if the active plot is part of a linked plot group. This is mainly because the velocity vector might not be present in all plots belonging to the group and the extensibility of the `create_streamlines`, `extract_streamlines`, and `set_streamlines_prop` commands cannot be ensured. However, if you want to apply streamlines to a plot group, you must create a special linked plot group to allow streamlines for that plot group (see Linking Plots on page 18 and link_plots on page 216).

NOTE If plot group members do not have similar data, the results might be unexpected.

Displaying Streamlines

Click the toolbar button to display the Streamlines dialog box (see Figure 112), where you can select the vector field, the starting point, and the display properties.

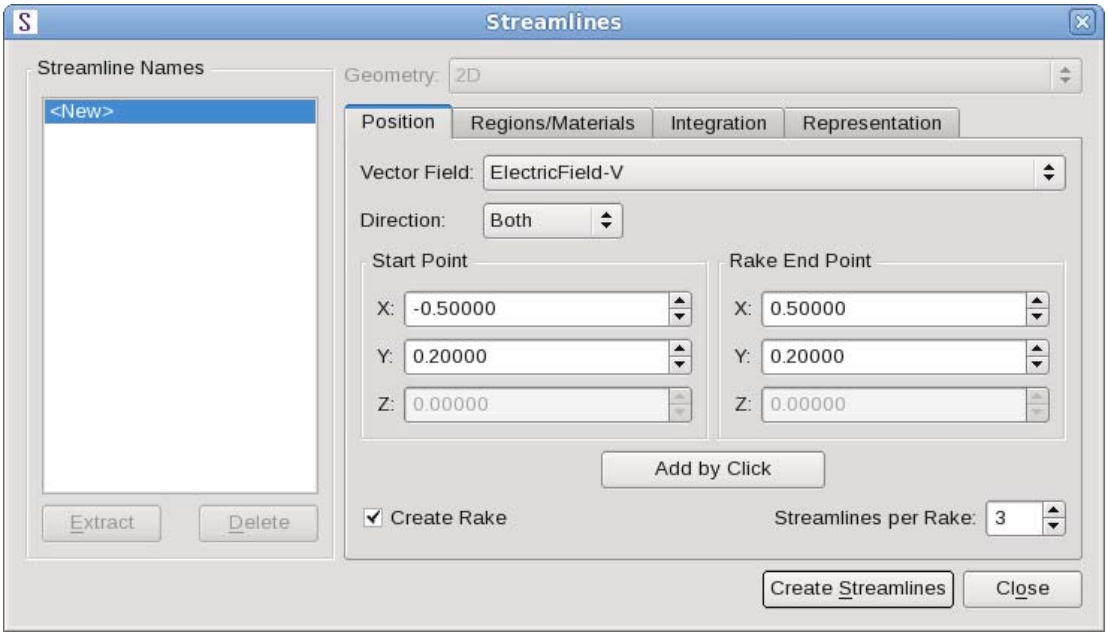


Figure 112 Streamlines dialog box showing Position tab, before creating streamlines

In this dialog box, several properties can be defined to customize the display of the streamlines.

Position Tab

The fields of the **Position** tab are:

- The **Vector Field** box is where you select the field used to calculate the streamlines.
- The **Direction** box allows you to show only streamlines ending on a point, starting from a point, or both.
- The **Create Rake** option allows you to create multiple streamlines between the start and end points. The number of streamlines is defined by the value in the **Streamlines per Rake** box.
- The **Add by Click** button allows you to add the start point and the endpoint for the rake using the mouse to click the selected plot. If the **Create Rake** option is selected, you can add two positions. If the **Create Rake** option is not selected, you can add one position.
- The **Create Streamlines** button allows you to create a family of streamlines going from a starting point to a rake end point. This button also changes its behavior when existing streamlines are selected from the list, allowing you to modify their attributes without creating new streamlines.

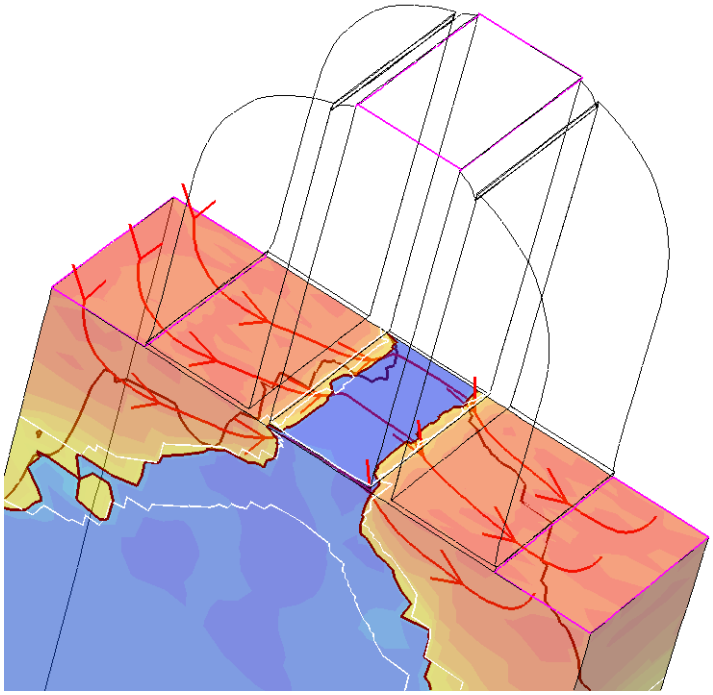


Figure 113 Example of displaying streamlines on plot

Specifying Regions or Materials

On the **Regions/Materials** tab, you can specify in which regions or materials the streamlines will be plotted. Not selecting a region or material causes Sentaurus Visual to plot the streamlines over the complete structure.

Representing the Streamlines

On the **Representation** tab, you can make cosmetic changes to the appearance of streamlines such as the line style, width, and resolution, as well as the color and the size of the vector field arrows.

Integration Settings

Sentaurus Visual has default integration settings that work with most of the simulation results obtained from other TCAD tools. However, users have the opportunity to fine-tune these values if needed. The **Integration** tab is for this purpose.

Integration Tab

By default, the Runge-Kutta 4 (RK4) algorithm is used for numeric integration of the fields. Some details about this integration can be modified. These values are included in the `create_streamline` Tcl command (see create_streamline on page 164). When you use the Streamlines dialog box, you can select between the values calculated by Sentaurus Visual or the default values specified in the User Preferences.

Step options are:

- **Initial** sets the initial step for the vector field integration. In the Runge-Kutta 4 (RK4) algorithm, the initial step is also a constant length for all steps.
- **Max Step** sets the maximum number of steps until the end of the integration. For termination constraints, either the **Max Step** value or the **Terminal Speed** value can be changed.

Others options are:

- **Maximum Propagation** controls the length of the streamline. If the **Both Direction** option is selected, the maximum length will be two times this value.
- **Terminal Speed** sets an end constraint for the numeric integration. If the particle speed is reduced to a value less than this number, the integration will end. For termination constraints, either the **Terminal Speed** value or the **Max Step** value can be changed.

Managing Created Streamlines

The Streamlines dialog box includes a list of the created streamlines in the Streamline Names pane. If you select **<New>** in this pane, you enter the creation mode, in which you can create streamlines. In the same way, if you select another streamline, the update mode is activated.

In the update mode, the Streamlines dialog box executes the `set_streamline_prop` Tcl command, which changes the representation of the streamlines by updating their properties without creating new ones in a faster way. Selecting a streamline in the Streamline Names pane also highlights the streamline in the plot, allowing you to easily identify the active streamline.

Configuring General Parameters of Streamlines

In the User Preferences dialog box (**Edit** > **Preferences**), several parameters are available that can be changed to improve the performance of creating streamlines (select **2D/3D** > **Streamlines**) (see Figure 114).

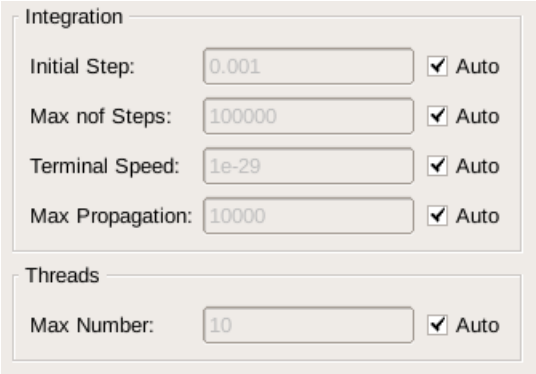


Figure 114 Parameters available for streamlines in User Preferences dialog box

Sentaurus Visual calculates the *best* integration parameters depending on the selected structure and vector fields. By default, it uses 10 threads to create rakes of streamlines. However, you can disable this behavior by clearing the **Auto** option next to the **Max Number** field (see Figure 114) and specifying another value.

Extracting Data From Streamlines

Sentaurus Visual can extract data from existing streamlines. Each streamline generates its own 1D dataset that contains the coordinates data defining the streamline as well as all scalar fields defined in the geometry.

Extracting data using the Streamlines dialog box will create a new xy plot (if it is not created already) and one curve for each streamline extracted, displaying the current contour-banded field (from the source plot) versus distance. If the field is scaled with something other than **Linear** or **Custom**, Sentaurus Visual will set the vertical axis (left y-axis) to logarithmic scale (see Visualizing Fields on page 60).

To extract data from streamlines:

1. Select one or more streamlines from the **Streamline Names** pane.

   This will make the **Extract** and **Delete** buttons available (see Figure 115).

2. Click **Extract**.

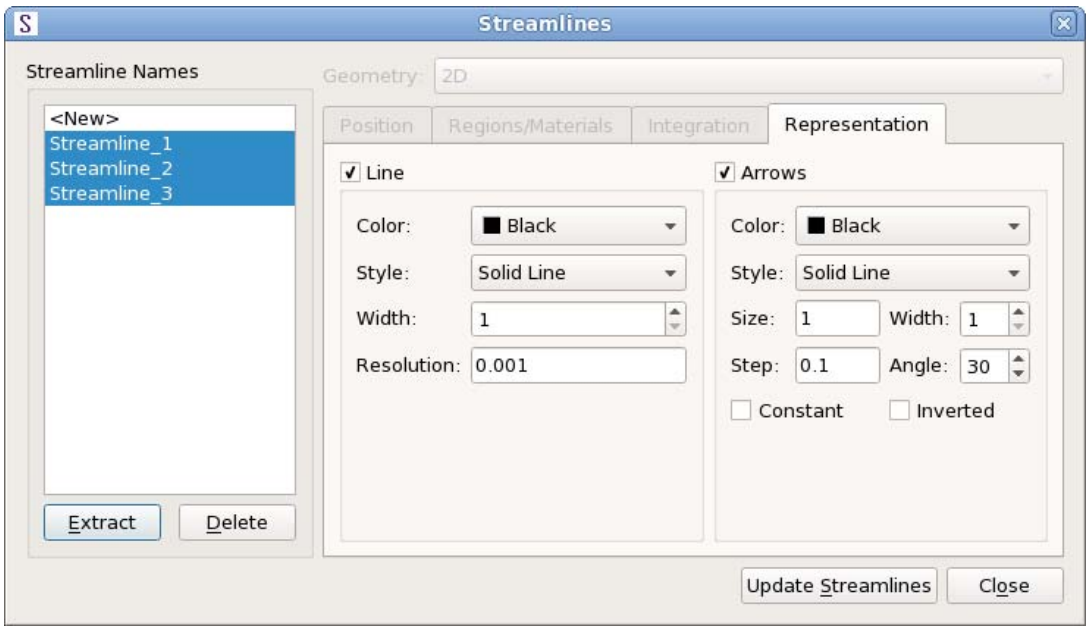


Figure 115 Streamlines dialog box showing Representation tab: streamlines have been created and some are selected, and the Extract button is available

Figure 116 on page 138 shows the results of the extraction operation.

The equivalent command for extracting data from streamlines is `extract_streamlines` (see extract_streamlines on page 181).

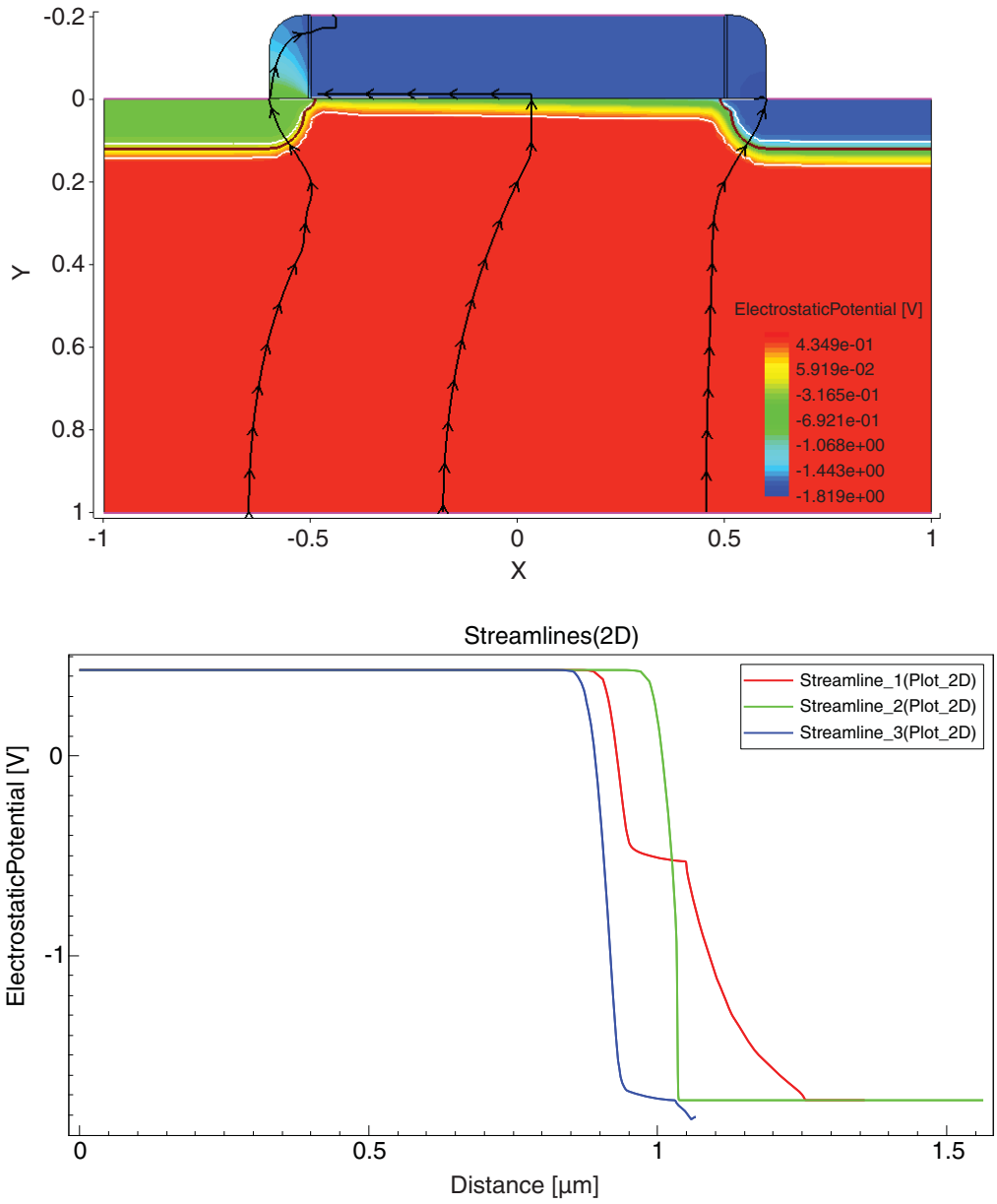


Figure 116 (*Top*) Two-dimensional plot displaying three streamlines and (*bottom*) xy plot displaying three curves from the data extracted from the streamlines

CHAPTER 6 Automated Tasks

This chapter presents how to automate tasks with Tcl scripting and Inspect compatibility.

Running Tcl Scripts

Sentaurus Visual allows scripts to be called from the command line or the GUI.

To run a script from the command line, type:

```
svisual /path/to/script.tcl
```

To run a script from the GUI:

- Choose **File** > **Run Tcl Script**.

Typical Uses of Tcl Scripts

The following examples illustrate some typical scripting uses in the context of batch scripts.

Example 1: Plotting Id–Vg Curve

The contents of the script `plot_idvg.tcl` are:

```
# Load PLT data file.
set mydata [load_file IdVg_n62_des.plt]

# Create new empty xy plot.
set myplot [create_plot -1d]

# Create Id-Vg curve using loaded dataset and display on new xy plot.
set IdVgcurve [create_curve -plot $myplot -dataset $mydata \
   -axisX "gate InnerVoltage" -axisY "drain TotalCurrent"]

# Customize the curve.
set_curve_prop $IdVgcurve -plot $myplot -show_markers -markers_size 7 \
   -color red -label "nMOS"
```

```
# Display grid and set grid properties.
set_plot_prop -show_grid
set_grid_prop -show_minor_lines \
   -line1_style dash -line1_color #a0a0a4 \
   -line2_style dot -line2_color #c0c0c0

# Assign axis labels and set range.
set_axis_prop -plot $myplot -axis x -title "Vgate (V)"
set_axis_prop -plot $myplot -axis y -title "Idrain (A/um)" -type log
set_axis_prop -plot $myplot -axis y -range {1e-09 0.0002}

# Export plot into PNG file.
export_view "curve.png" -plots $myplot -resolution 500x500 -format PNG \
   -overwrite
```

The first three commands of this script open a `.plt` file and create an I_d–V_g curve. Next, the plot is customized to make it more readable. Finally, the plot is exported to a `.png` file (see Figure 117).

NOTE This script must be executed with the virtual X server option to allow graphics export in batch mode, for example:

```
% svisual -batchx plot_idvg.tcl
```

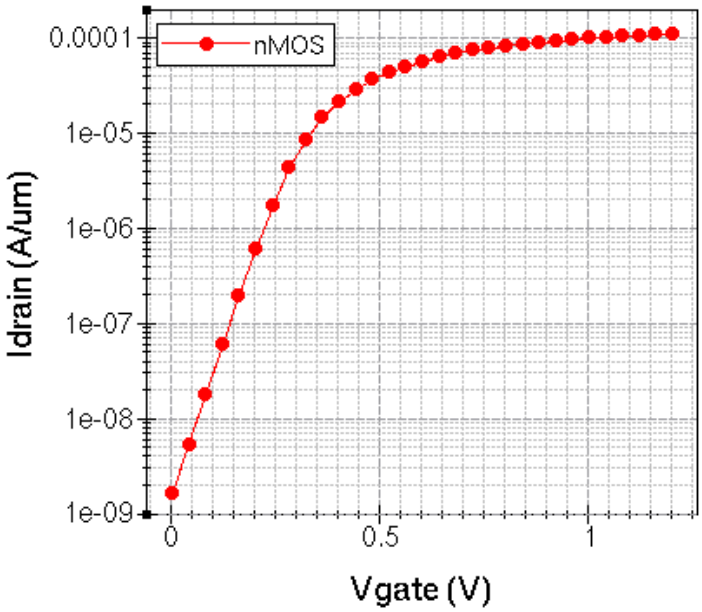


Figure 117 I_d–V_g plot from xy data

Example 2: Creating a Cutline and Exporting Cutline Data to CSV File for Further Processing

The contents of the script `plot_npn.tcl` are:

```
# Load TDR file.
set mydata2D [load_file npn_msh.tdr]

# Create new plot.
set myplot2D [create_plot -dataset $mydata2D]

# Create 1D cutline normal to x-axis at point x=-0.005.
set mydata1D [create_cutline -plot $myplot2D -type x -at -0.005]

export_variables {DopingConcentration xMoleFraction Y} \
   -dataset $mydata1D -filename "data.csv" -overwrite
```

The first two commands load and display a TDR file. The next `create_cutline` command creates a cutline at the specified location. The last command exports the selected variables from the cutline to a CSV file.

NOTE This script can be run solely in batch mode, with the command:

```
% svisual -batch plot_npn.tcl
```

Saving Command History

Almost every action performed in Sentaurus Visual is replicated in the Tcl Command panel. These actions can be saved to be executed in another session by clicking the **Save** button of the Tcl Command panel.

Running Inspect Command Files

Sentaurus Visual can run Inspect command files.

You can run an Inspect command file in the same way as for a native Sentaurus Visual Tcl script.

Script Library

Sentaurus Visual allows you to add Tcl script files as libraries, which can be loaded automatically at startup or manually using the Tcl command `load_library` (see load_library on page 238).

A script library has the file name formatted as `<libraryName>.tcl`.

The default library path is `$STROOT_LIB/svisuallib`. In addition, it includes a user-defined library path, which is set by default to `${HOME}/svisuallib`, but it can be modified in the user preferences.

NOTE The default value for the `STROOT_LIB` variable is:

```
$STROOT/tcad/$STRELEASE/lib
```

Both paths can be checked for Tcl scripts (any file with the extension `.tcl`) for auto-loading at startup, which can be enabled or disabled in the user preferences.

The options related to launching Sentaurus Visual are only valid when the auto-loading of the script library is enabled:

`-nolibrary`	Disables the auto-loading of scripts from the library.
`-library_path <customPath>`	Adds a custom path to the list of library paths to look for script files when auto-loading is enabled.

Restrictions

Every procedure defined in a script library must begin with the prefix `lib_` to avoid the possible redefining of any existing Sentaurus Visual command.

At the time of loading one or more script files from the script library paths, if there are procedures that have been defined without this prefix, a warning message will be displayed, listing these procedures.

Moreover, if there are procedures that redefine Sentaurus Visual commands, a second warning message is displayed.

APPENDIX A Tcl Commands

This appendix describes the tool command language (Tcl) commands that can be used in Sentaurus Visual.

The Tcl commands apply to all plots and structures unless stated otherwise.

Syntax Conventions

The following conventions are used for the syntax of Tcl commands:

- Angle brackets – <> – indicate text that must be replaced, but they are *not* part of the syntax.
- Braces – { } – are used for lists of values, and they must be included in the syntax.
- Brackets – [] – indicate that the argument is optional, but they are *not* part of the syntax.
- Parentheses – () – are used solely to group arguments to improve legibility of commands, but they are *not* part of the syntax.
- A vertical bar – | – indicates options, only one of which can be specified.

Object Names: -name Argument

For all Tcl commands that use the `-name` argument, if a name conflict is detected, Sentaurus Visual will print an error message and stop execution of the command, for example:

```
create_plot -name newPlot -dataset 3D
#-> newPlot

create_plot -name newPlot -dataset 2D
#-> "Error: create_plot: The plot couldn't be created. Plot name 'newPlot'
already exists."
```

If you do not specify the `-name` argument in a command, Sentaurus Visual will generate an internal name that will remain consistent in a script. If the name generated conflicts with a name defined later in a script for the same type of element (such as a curve or cutline), Sentaurus Visual will print an error message and stop execution of the command.

Common Properties

The following properties are used in several Tcl commands.

Colors

In Tcl commands that allow you to specify color properties (such as the `-color <#rrggbb>` option), a string specifying red, green, and blue components of the RGB system is expected. The string is preceded by a hash (#) character, and each value is provided in hexadecimal form. Common colors also have aliases as listed in Table 9.

Table 9 Common colors

Alias	General form	Description
`white`	`#ffffff`	White
`black`	`#000000`	Black
`red`	`#ff0000`	Red
`darkRed`	`#800000`	Dark red
`green`	`#00ff00`	Green
`darkGreen`	`#008000`	Dark green
`blue`	`#0000ff`	Blue
`darkBlue`	`#000080`	Dark blue
`cyan`	`#00ffff`	Cyan
`darkCyan`	`#008080`	Dark cyan
`magenta`	`#ff00ff`	Magenta
`darkMagenta`	`#800080`	Dark magenta
`yellow`	`#ffff00`	Yellow
`olive`	`#808000`	Olive
`gray`	`#a0a0a4`	Gray
`darkGray`	`#808080`	Dark gray
`lightGray`	`#c0c0c0`	Light gray

Fonts

For Tcl commands that allow you to adjust font properties, Sentaurus Visual defines a specific list of font families and attributes listed in Table 10.

Table 10 Font families and their attributes

Font family	Attribute
Arial Courier Times	Bold Italic Normal Strikeout Underline

NOTE In xy plots, the font size of different elements of the plot are set with the `font_size` argument; whereas in 2D and 3D plots, the font size cannot be set directly. Instead, the font size is set as a factor of the plot frame (the default value is 1.0), with the `font_factor` argument.

Lines

For Tcl commands that allow you to adjust line properties (such as the `-line_style` option), Sentaurus Visual defines a specific list of line styles listed in Table 11. You can provide the name of the style or its short form directly.

Table 11 Line styles

Name of line style	Short form of line style	Description
`solid`	_	Continuous line: _____
`dot`	.	Dotted line: .....
`dash`	-	Dashed line: ----------
`dashdot`	`-.`	Alternating dash-and-dot line: -.-.-.-.-.-.-
`dashdotdot`	`-..`	Alternating dash-and-two-dots line: -..-..-..-..-

Markers

Table 12 lists the different markers available to use in xy plots in Sentaurus Visual. Tcl commands allow you to use the name or the short form of each marker.

Table 12 Marker types

Name of marker type	Short form of marker type	Description
`circle`	`o`	○
`circlef`	`of`	●
`diamond`		◇
`diamondf`		◆
`square`		□
`squaref`		■
`plus`	`+`	+
`cross`	`x`	x

add_custom_button

Adds a custom button to the Scripts toolbar.

NOTE This command works only in interactive mode. In batch mode, this command has no effect.

NOTE If both `-icon` and `-name` are specified, only the icon will be shown in the toolbar.

Syntax

```
add_custom_button
   (-file <stringValue> | -script <stringValue>)
   [-desc <stringValue>] [-icon <stringValue>] [-label <stringValue>]
   [-name <stringValue>] [-separator]
```

Argument	Description
`-desc <stringValue>`	Text that describes the button. This appears as the tooltip of the button.
`-file <stringValue>`	Name of a Tcl file to execute.
`-icon <stringValue>`	Specifies a graphics file to be used as the button icon. Supported file formats are BMP, GIF, PNG, and SVG.
`-label <stringValue>`	Specifies a label for the button.
`-name <stringValue>`	Name of the button in the Scripts toolbar. See Object Names: -name Argument on page 143.
`-script <stringValue>`	Name of a Tcl script to execute.
`-separator`	Adds a separator between buttons in the Scripts toolbar.

Returns

String with the name of the custom button specified by `-name`. If this argument is not specified, the command returns the default name, which is `S<number>`, where `<number>` starts at 1 and increases each time a custom button is created.

Example

```
add_custom_button -script "echo Hello World!" -name Greetings \
   -desc "Echoes a Hello World Message."
#-> Greetings
```

add_frame

Adds a new frame to the frame buffer.

NOTE You must use of the `start_movie` command before using the `add_frame` command.

Syntax

```
add_frame [-name <frameName>] [-plot <plotName>]
```

Argument	Description
`-name <frameName>`	Name of the new frame to be captured. See Object Names: -name Argument on page 143.
`-plot <plotName>`	Name of the plot where the new frame will be saved. Current active plot is used by default.

Returns

String.

Example

```
add_frame -name Frame1
#-> Frame1
```

See Also

start_movie on page 298

calculate

This function extracts FET parameters from I_d–V_d or I_d–V_g curves.

NOTE This command applies to xy plots only.

Syntax

```
calculate <curveName> [-plot <plotName>]
   -op (vth | gmmax | idsat | ioff | rout | ron)
```

Argument	Description
`<curveName>`	Name of the curve on which to apply the parameter extraction.
`-plot <plotName>`	Name of the plot on which to apply the parameter extraction.
`-op vth | gmmax | idsat | ioff | rout | ron`	Parameter to be extracted from the curve. For more detailed information about the extraction parameters, see Analysis Tool on page 56.

Returns

Double.

Example

```
calculate Curve_1 -op ron
#-> 0.0554013
```

calculate_field_value

Calculates the minimum and maximum values of a particular field, and shows (with a marker) the location of these values.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
calculate_field_value [-field <fieldName>] (-min | -max)
   [-geom <geometryName>] [-plot <plotName> | -dataset <dataName>]
   [-materials <materialsList> | -regions <regionsList>]
   [-ranges {<x1> <x2> <y1> <y2> [<z1> <z2>]}]
```

Argument	Description
`-field <fieldName>`	Name of the field from which the data will be obtained. If not specified, the command uses the current contour-band field displayed in the plot.
`-min | -max`	Selects whether the point returned will be the minimum or maximum value.
`-geom <geometryName>`	Selects the geometry from which the data will be extracted.
`-plot <plotName> | -dataset <dataName>`	Selects either the plot or the dataset (for batch mode) from which the data will be extracted.
`-materials <materialsList> | -regions <regionsList>`	Sets a list of materials or regions that will be used to find the minimum or maximum value.
`-ranges {<x1> <x2> <y1> <y2> [<z1> <z2>]}`	Sets a specific range of values to find the minimum or maximum value. After the command is executed, the minimum or maximum value and its position are returned.

Returns

Double and a list of coordinate values.

Example

```
calculate_field_value -plot Plot_n9_des -field Abs(TotalCurrentDensity-V)
   -min
```

```
# position:
#  min:  271.172 5.39062 0  value  0.0206237
#  max:  277   -0.546875 0  value  426.765
#-> 0.0206237318885 {271.172 5.39062 0}
```

calculate_scalar

Calculates a scalar value.

This command operates over curves or variables. If `-curves` is specified, `-plot` can specify a plot in memory. Otherwise, the command uses the curve in the selected plot.

NOTE This command applies to xy plots only.

Syntax

```
calculate_scalar (-curves [-plot <stringValue>] | -variables)
   -function <stringValue>
```

Argument	Description
`-curves | -variables`	Select whether the command parses the specified formula as a function of curves or variables.
`-function <stringValue>`	Specifies a formula to be used to extract a scalar value. All mathematical operations can be used; however, you must ensure that the last operation that encloses the entire set of functions is a scalar value function. Otherwise, the command will fail. See Appendix C on page 315 for information about which functions return a double value (scalars).
`-plot <sringValue>`	Name of the plot where the command will search for curves. If not specified, the command uses the selected xy plot.

Returns

Double.

Example

```
calculate_scalar -curves -plot Plot_1 \
   -function vecmax(<Curve_1>)+vecmin(<Curve_1>)
#-> 0.000351277048757
```

create_curve

Creates a new curve for an xy plot.

If `-plot` is not specified, the command draws the curve on the selected plot. If there are no xy plots created or the selected plot is not an xy plot, the command returns an error.

NOTE This command applies to xy plots only.

Syntax

```
create_curve [-name <curveName>] [-plot <plotName>]
(
   (-dataset {<dataNamesList>} -axisX <varX> (-axisY | -axisY2) <varY>) |
   -function <formula>
)
```

Argument	Description
`-name <curveName>`	Name of the new curve. If not specified, the command assigns a default curve name. See Object Names: -name Argument on page 143.
`-plot <plotName>`	Name of the plot where the new curve will be displayed. If not specified, the command draws the curve on the selected xy plot.
`-dataset {<dataNamesList>}`	List of dataset names where the information is extracted.
`-axisX <varX>`	Specifies the variable to be used for the x-axis.
`-axisY | -axisY2 <varY>`	Specifies the variable to be used for the y-axis or the y2-axis.
`-function <formula>`	Specifies a formula from which to create a curve.

Returns

List.

Example

```
create_curve -plot Plot_1 -dataset IdVd_example -axisX "drain OuterVoltage"
   -axisY "drain TotalCurrent"
#-> Curve_1
```

create_cut_boundary

Creates a cutline along the specified structure boundary.

You must select the regions or materials associated with the boundary for avoid the interface between regions becoming ambiguous.

The command produces a list of line segments that define the cutline. A segment is defined as the union of two vertices, where a vertex is a point that defines a region (angle > 30° with its neighboring points).

If `-plot` or `-dataset` is not specified, the command uses the selected 2D plot dataset.

NOTE This command applies to 2D plots only.

Syntax

```
create_cut_boundary -points <pointsList>
   (-materials <materialsList> | -regions <regionsList>)
   [-name <cutName>] [-plot <plotName> | -dataset <dataName>]
   [-reverse] [-segments <stringList>]
```

Argument	Description
`-points <pointsList>`	Vertices on the boundary through which the cutline will pass. Points must be defined as: `{"x0 y0 [z0]" "x1 y1 [z1]" ...}` Any component not present in the plot must be set to 0.
`-materials <materialsList> | -regions <regionsList>`	Materials or regions to which the boundary belongs.
`-name <cutName>`	Name of the cutline dataset. If not specified, the command generates a default name. See Object Names: -name Argument on page 143.
`-plot <plotName> | -dataset <dataName>`	Specifies the plot or dataset from which to retrieve the regions or materials. If not specified, the command uses the selected plot.
`-reverse`	Reverses the direction of the cutline creation backwards.
`-segments <stringList>`	Specifies the regions from which the data will be extracted in each segment line. The length of the list must be exactly the number of vertices along the cutline minus 1.

Returns

String (the name of the resultant 1D dataset).

Example

```
create_cut_boundary -plot Plot_2D -regions {R.Gateox R.PolyReox}
   -segments {R.Gateox R.PolyReox}
   -points {"-0.5125 -0.002 0" "-0.6 -0.002 0" "-0.6 0 0"} -name myCut

#->myCut
```

create_cutline

Creates a new cutline.

If the type of cutline is aligned to an axis, you must specify the `-at` argument.

If `-type free` is specified, you must specify the `-points` argument. The new plot created has the name of the cutline.

NOTE This command applies to 2D and 3D plots only. Only 3D plots accept `-type free` cutlines.

Syntax

```
create_cutline -type (x | y | z | free)
   [-at <doubleValue> | -points {<x1> <y1> [<z1>] <x2> <y2> [<z2>]}]
   [-dataset <dataName> | -plot <plotName>]
   [-materials <stringList> | -regions <stringList>] [-name <cutlineName>]
```

Argument	Description
`-at <doubleValue> | -points {<x1> <y1> [<z1>] <x2> <y2> [<z2>]}`	If an axis is selected using `-type`, the `-at` argument must be used. If `-type free` is specified, two (x,y) points must be specified with the `-points` argument. (In 3D plots, you must specify two (x, y, z) points.)
`-dataset <dataName> | -plot <plotName>`	Name of the dataset or plot from where the cutline will be generated. If neither is specified, the command uses the selected 2D or 3D plot dataset.
`-materials <stringList> | -regions <stringList>`	If specified, the cutline operation is performed only on the materials or regions listed.
`-name <cutlineName>`	Name of the new cutline dataset. If not specified, the command generates a default name. See Object Names: -name Argument on page 143.
`-type x | y | z | free`	Selecting `x`, `y`, or `z` ties the cutline to the specified axis. The `free` option allows you to create a cutline drawing a line between two (x,y) coordinates.

Returns

String (the name of the resultant 1D dataset).

Example

```
create_cutline -plot Plot_2D -type free -points {-0.45 -0.15 0.30 0.80}
   -name myCutlineDataset
#-> myCutlineDataset
```

create_cutplane

Creates a new cutplane.

The new plot created has the name of the cutplane. If `-plot` or `-dataset` is not specified, the command uses the selected 3D plot dataset.

NOTE This command applies to 3D plots only.

Syntax

```
create_cutplane -type (x | y | z | free)
   [-at <doubleValue> | (-origin {<x> <y> <z>} -normal {<x> <y> <z>})]
   [-dataset <dataName> | -plot <plotName>] [-name <cutplaneName>]
```

Argument	Description
`-at <doubleValue>	-origin {<x> <y> <z>} -normal {<x> <y> <z>}`	With `-at`, cuts the structure at the value specified in the axis defined by `-type` (it must not be `free`). With `-origin` and `-normal`, cuts the structure with a plane defined by the given origin and normal. The argument `-type` must be `free`.
`-dataset <dataName>	-plot <plotName>`	Name of the dataset or plot from where the cutplane will be generated. If not specified, the command uses the selected plot.
`-name <cutplaneName>`	Name of the new cutplane dataset. If not specified, the command generates a default name as a function of the original 3D dataset. See Object Names: -name Argument on page 143.
`-type x	y	z	free`	Selects the axis from which the cutplane will be generated.

Returns

String (the name of the resultant 2D dataset).

Example

```
create_cutplane -plot Plot_3D -name Cut1 -type y -at 0.3
#-> Cut1
```

create_cutpolyline

Creates a new cutline with the specified number of vertex points.

NOTE This command applies to 2D plots only.

Syntax

```
create_cutpolyline -points <pointsList>
   [-dataset <dataName> | -plot <plotName>]
   [-materials <stringList> | -regions <stringList>] [-name <cutName>]
```

Argument	Description
`-dataset <dataName> | -plot <plotName>`	Name of the dataset or plot from where the cutline will be created. If neither is specified, the command uses the selected 2D plot dataset.
`-materials <stringList> | -regions <stringList>`	If specified, the cutline will be performed only on the materials or regions listed.
`-name <cutName>`	Name of the cutline. See Object Names: -name Argument on page 143.
`-points <pointsList>`	Points in the region where the cutline will pass through. Points must be defined as `{"x0 y0" "x1 y1" ...}`.

Returns

String.

Example

```
create_cutpolyline -plot Plot_2D
   -points {"0.05872 -0.260434" "0.46536 -0.034674" "0.26 0.074126"}

#-> C1(2D)
```

create_field

Creates a new field using data from the plot or the dataset specified in the arguments.

The command uses the selected 2D or 3D dataset if `-plot` or `-dataset` is not specified.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
create_field -function <functionToEvaluate> -name <fieldName>
   [-plot <plotName> [-geom <geometryName>] | -dataset <dataName>]
   [-show]
```

Argument	Description
`-function <functionToEvaluate>`	Specifies the function to be evaluated. For a complete list of operations, see Appendix C on page 315.
`-name <fieldName>`	Name of the new field. See Object Names: -name Argument on page 143.

`-plot <plotName>`
`[-geom <geometryName>] |`
`-dataset <dataName>`
	Name of the plot or dataset from where the field is created. If not specified, the command uses the active plot.
	The argument `-geom` can be used only with `-plot`. It specifies the name of the geometry in the plot. If not specified, the command uses the first geometry associated with the plot.

`-show`	Shows immediately the newly created field if specified.

Returns

String.

Example

```
create_field -name newFld -dataset 3D -function "log(<ElectricField>)"
#-> newFld
```

create_iso

Creates a new iso-geometry using an isovalue from the field of a geometry, or modifies an existing iso-geometry.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
create_iso [-plot <plotName>] [-geom <geometryName>] [-field <fieldName>]
   [-value <floatValue>] [-name <datasetName> | -modify] [-color <color>]
```

Argument	Description
`-plot <plotName>`	Specifies the plot from which to retrieve the geometry. If not specified, the command uses the selected plot.
`-geom <geometryName>`	Name of the geometry in the plot. If not specified, the command uses the first geometry associated with the plot. If `-modify` is specified, this argument is the name of the iso-geometry to be modified.
`-field <fieldName>`	Selects the field on which the isovalue is used. If not specified, the command uses the current contour band field displayed in the plot.
`-value <floatValue>`	Specifies the isovalue with which to create the new iso-geometry.
`-name <datasetName> | -modify`	Specifies either the name of the new iso-geometry to be created or that the geometry defined by `-geom` will be modified by the arguments in the command. See Object Names: -name Argument on page 143.
`-color <color>`	Specifies the color to be used for the new iso-geometry. If not specified, the command uses the default color (gray).

Returns

String naming the new geometry or the modified geometry.

Example

```
create_iso -plot Plot_3D -geom 3D -field ElectrostaticPotential -value 0.0
   -name Iso1(3D) -color blue
#-> Iso1(3D)

create_iso -modify -geom Iso1(3D) -value 0.8 -color green
#-> Iso1(3D)
```

create_plot

Creates an empty xy plot, or creates a plot from 2D or 3D datasets.

Syntax

```
create_plot
(
   -1d | (-1d -dataset <dataName>) | -dataset <dataName> |
    -duplicate <plotName>
)
   [-name <plotName>] [-ref_plot <plotName>] [-tdr_state_index <intValue>]
```

Argument	Description
`-1d | (-1d -dataset <dataName>) | -dataset <dataName> | -duplicate <plotName>`	The arguments are: • `-1d` creates an empty xy plot. • `-dataset` creates a plot from a loaded 1D, 2D, or 3D dataset. • `-duplicate` replicates the properties of the plot in a new plot (applies to xy plots only).
`-name <plotName>`	Name of the new plot. See Object Names: -name Argument on page 143.
`-ref_plot <plotName>`	Name of the plot to be used as a reference to inherit the fields and the region properties. This argument applies to 2D and 3D plots only.
`-tdr_state_index <intValue>`	The new plot will load only the specified TDR state index from an already loaded dataset. The resulting plot is not considered be to a multistate plot. This argument applies to 2D and 3D plots only.

Returns

String.

Example

```
create_plot -dataset 3D
#-> Plot_3D
```

create_projection

Creates a 2D plot with maximum or minimum values along a 3D plot axis.

The argument `-resolution` increases the precision of the maximum or minimum calculation. Higher values of resolution lead to longer calculation times.

NOTE This command applies to 3D plots only.

Syntax

```
create_projection -field <fieldName> -function (min | max) -normal (x | y | z)
   [-geom <geometryName>] [-name <plotName>]
   [-plot <plotName> | -dataset <dataName>] [-resolution <x>x<y>x<z>]
   [-window {<x1> <y1> [<z1>] <x2> <y2> [<z2>]} | -regions <regionsList> |
   -materials <materialsList>]
```

Argument	Description
`-field <fieldName>`	Name of the field to be projected.
`-function (min | max)`	Specifies either the maximum values projection or minimum values projection.
`-normal (x | y | z)`	Specifies the axis to be projected: • `x` projects a 2D yz plot. • `y` projects a 2D xz plot. • `z` projects a 2D xy plot.
`-geom <geometryName>`	Geometry that has the given field. If not specified, the command uses the first shown geometry from the selected dataset.
`-name <plotName>`	Name of the resultant 2D projection plot. See Object Names: -name Argument on page 143.
`-plot <plotName> |` `-dataset <dataName>`	Specifies the plot or dataset from which to retrieve the regions or materials. If not specified, the command uses the selected plot.
`-resolution <x>x<y>x<z>`	Specifies the resolution of the maximum or minimum search data algorithm. If not specified, `-resolution 50x50x50` is used by default.

`-window {<x1> <y1> [<z1>]`
`<x2> <y2> [<z2>]} |`
`-regions <regionsList> |`
`-materials <materialsList>`
	Materials or regions where maximum or minimum data will be extracted. If `-window` is used, all regions inside that window are selected. If none of these options is specified, the command uses all regions in the entire domain.

Returns

String.

Example

```
create_projection -Plot_3D -field DopingConcentration -function max -normal x
   -resolution 50x50x50
```

```
#-> Projection_max(3D)
```

create_streamline

Creates a new streamline on a 2D or 3D plot.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
create_streamline -field <fieldName>
(
   -point {<x> <y> [<z>]} |
   -p1 {<x> <y> [<z>]} -p2 {<x> <y> [<z>]} -nofpoints <intValue>
)
   -direction (backward | both | forward)
   [-geom <geometryName>]
   [-integ_initial_step <doubleValue>]
   [-integ_max_propagation <doubleValue>]
   [-integ_max_steps <intValue>]
   [-integ_terminal_speed <doubleValue>]
   [-materials <materialsList> | -regions <regionsList>]
   [-name <streamlineName>] [-plot <plotName>]
```

Argument	Description
`-field <fieldName>`	Selects the field on which the streamline will be created.

`-point {<x> <y> [<z>]} |`
`-p1 {<x> <y> [<z>]}`
`-p2 {<x> <y> [<z>]}`
`-nofpoints <intValue>`
	Use the `-point` argument to create one streamline:
	• If the direction is `forward`, the point specified is the starting point of the streamline.
	• If the direction is `backward`, the point specified is the end point of the streamline.
	• If the direction is `both`, the point specified is the middle of the streamline.
	Use the `-nofpoints` argument to create a custom number of streamlines going from point 1 to point 2 with the `-p1` and `-p2` arguments. For example, if you specify `-nofpoints 9`, then 7 streamlines in addition to the streamlines originating from point 1 and point 2 will be created. Analogous to the `-point` argument, the direction determines the type of point.

`-direction backward	both	forward`	Direction of the streamline created. Default: `both`.
`-geom <geometryName>`	Name of the geometry in the plot. If not specified, the command uses the first geometry associated with the plot.
`-integ_initial_step <doubleValue>`	Specifies the initial step used in all the calculations of the Runge-Kutta 4 algorithm. This step remains unchanged in all the mathematical processes.

`-integ_max_propagation <doubleValue>`	Maximum propagation is the maximum length of the streamline after calculating the magnitude of each step iteration. This is related to the 'line' length at the calculation level. This represents the maximum amount of data that will be calculated to generate a line of 'x'-length. However, you can represent the line with less data using the `-line_resolution` argument of the command `set_streamline_prop`. The value of `-line_resolution` must *not* be less than the value of `-integ_max_propagation`. Otherwise, it can lead to unexpected results.
`-integ_max_steps <intValue>`	Specifies the number of iterations of the Runge-Kutta 4 algorithm to be applied.
`-integ_terminal_speed <doubleValue>`	For each iteration, the Runge-Kutta 4 algorithm calculates the differential value of the vector field in the point and, using the Picard–Lindelöf iteration method, this value is equal to the speed value of the field in that point. This argument specifies the 'terminal speed' of the algorithm. If the next result iteration of the Runge-Kutta 4 algorithm is lower than this value, the integration stops.
`-materials <materialsList> | -regions <regionsList>`	Specify either: • Materials on which the streamline will be created. Default: All materials. • Region on which the streamline will be created. Default: All regions.
`-name <streamlineName>`	Identifier for the new streamline created. If not specified, the command generates a default name. See Object Names: -name Argument on page 143.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

List.

Example

```
create_streamline -field ElectricField -point {0.5 0.2} -direction both
#-> Streamline_1
```

create_surface

Creates a new 3D dataset using a 2D dataset or geometry of a plot as source, or modifies an existing 3D surface dataset.

NOTE This command applies to 2D datasets (or plots) and 3D surface datasets (or plots) only.

Syntax

```
create_surface
   [-dataset <datasetName> | -plot <plotName> [-geom <geometryName>]]
   [-factor <floatValue>] [-field <fieldName>] [-name <datasetName>]
   [-range <doubleList>] [-scale (linear | logabs | asinh)]
```

Argument	Description
`-dataset <datasetName>`	Specifies the dataset to be used to build the surface dataset.
`-factor <floatValue>`	Specifies the factor that is used to multiply the coordinate to generate the surface. If not specified, the command uses one (1).
`-field <fieldName>`	Selects the field on which the surface is used. If not specified, the command uses the current contour band field displayed in the plot.
`-geom <geometryName>`	Name of the geometry in the plot. If not specified, the command uses the first geometry associated with the plot.
`-name <datasetName>`	Name of the surface dataset to be created. See Object Names: -name Argument on page 143.
`-plot <plotName>`	Specifies the plot from which to retrieve the geometry. If the plot is not specified, the command uses the selected plot.
`-range <doubleList>`	Pair of values (min and max) that defines the range to be used to limit the new coordinate values that generate the surface. If not specified, the command uses the entire range of the field.
`-scale linear | logabs | asinh`	Specifies the scale to be used on the coordinates to generate the surface dataset. If not specified, the command uses the linear scale.

Returns

String naming the new dataset.

Example

```
create_surface -plot Plot_2D -geom 2D -field ElectrostaticPotential
   -factor 0.2 -name Surface1(2D)
#-> Surface1(2D)
```

create_variable

Creates a new variable.

NOTE This command applies to xy plots only.

Syntax

```
create_variable -dataset <dataName> -name <varName>
   (-function <functionToEvaluate> | -values {<numericList>})
```

Argument	Description
`-dataset <dataName>`	Dataset from which values are obtained to evaluate functions.
`-function <functionToEvaluate> | -values {<numericList>}`	Expression to evaluate or the list of values to add to the dataset specified.
`-name <varName>`	Name of the new variable. See Object Names: -name Argument on page 143.

Returns

String.

Example

```
create_variable -name nVar -dataset idvd -values {0.1 0.3 0.5 0.7 0.9}
#-> nVar
```

diff_plots

Creates a new dataset with the difference in the common fields of the selected plots.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
diff_plots {<listOfPlots>} [-display] [-normalized]
```

Argument	Description
`{<listOfPlots>}`	List of plots to create the differential plot.
`-display`	Creates a new plot with the field difference dataset.
`-normalized`	Normalizes the values between the two plots.

Returns

String.

Example

```
diff_plots {Plot1 Plot2}
#-> Plot1-Plot2
```

draw_ellipse

Draws an ellipse at the specified position. The ellipse is represented in relation to the rectangle that envelops it, although the rectangle is not drawn.

NOTE This command applies to xy plots only.

Syntax

```
draw_ellipse -p1 {<x1> <y1>} -p2 {<x2> <y2>} [-plot <plotName>]
```

Argument	Description
`-p1 {<x1> <y1>}`	Specifies the upper-left corner of the rectangle that envelops the ellipse.
`-p2 {<x2> <y2>}`	Specifies the lower-right corner of the rectangle that envelops the ellipse.
`-plot <plotName>`	Name of the plot where the ellipse will be drawn. If not specified, the command uses the selected plot.

Returns

Returns a string naming the ellipse. The ellipse will be numbered if other ellipses are already drawn.

Example

```
draw_ellipse -plot Plot_XY -p1 {0 0.5} -p2 {0.75 0.25}
#-> Ellipse_1
```

draw_line

Draws a line connecting two points.

NOTE This command applies to xy and 2D plots only.

Syntax

```
draw_line -p1 <point1> -p2 <point2> [-plot <plotName>]
```

Argument	Description
`-p1 <point1>`	List of double values representing a point in the plot. This point is the first point of the line.
`-p2 <point2>`	List of double values representing a point in the plot. This point is the second point of the line.
`-plot <plotName>`	Name of the plot in which the line will be drawn.

Returns

Returns a string naming the line. The line will be numbered if other lines are already drawn.

Example

```
draw_line -plot Plot_n9_des -p1 {62.9161 49.8411} -p2 {138.117 60.5841}
#-> MakoVtkLine_1
```

draw_rectangle

Draws a rectangle in the current plot.

NOTE This command applies to xy and 2D plots only.

Syntax

```
draw_rectangle -p1 <point1> -p2 <point2> [-plot <plotName>]
```

Argument	Description
`-p1 <point1>`	List of double values representing the upper-left corner of the rectangle.
`-p2 <point2>`	List of double values representing the lower-right corner of the rectangle.
`-plot <plotName>`	Name of the plot in which the rectangle will be drawn.

Returns

Returns a string naming the rectangle. The rectangle will be numbered if other rectangles are already drawn.

Example

```
draw_rectangle -plot Plot_n9_des -p1 {46.0341 38.0749} -p2 {97.1916 112.253}
#-> Plane_1
```

draw_textbox

Draws a text box with a label at a position indicated by the `-at` argument, and inserts an arrow that points to the direction of the text box indicated by the `-anchor` argument.

NOTE This command applies to xy and 2D plots only. The arrow and its properties work only in 2D plots.

Syntax

```
draw_textbox -at <doubleList> -label <text>
   [-anchor <doubleList>] [-plot <plotName>]
```

Argument	Description
`-at <doubleList>`	List of two double values indicating the lower-right corner of the text box.
`-label <text>`	Specifies the text to be displayed in the text box.
`-anchor <doubleList>`	List of double values representing a position where the arrow will point to.
`-plot <plotName>`	Name of the plot in which the text box will be drawn.

Returns

Returns a string naming the text box. The text box will be numbered if other text boxes are already drawn.

Example

```
draw_textbox -at {219.458 41.6559} -anchor {219.458 41.1443} -label Text
#-> Text
```

echo

Prints a string in the Tcl Command panel.

Syntax

```
echo [<strings>]
```

Argument	Description
`<strings>`	String list to be printed in the Tcl Command panel.

Returns

None.

Example

```
echo "Hello World"
# Hello World
```

exit

Exits Sentaurus Visual with the status given as an argument.

Syntax

```
exit [<intValue>]
```

Argument	Description
`<intValue>`	Exit code as an integer. Default: `0`.

Returns

None.

Example

```
exit 1
# Exit status: 1
```

export_curves

Exports a curve to the specified file format.

Syntax

```
export_curves -filename <fileName> -plot <plotName>
   [{<curvesList>}] [-format (csv | plx)] [-overwrite]
```

Argument	Description
`-filename <fileName>`	Name of the exported file or files.
`-plot <plotName>`	Exports the curves from the specified plot.
`{<curvesList>}`	List of curves to be exported.
`-format csv | plx`	Format of the output file. Default is `csv`.
`-overwrite`	Overwrites existing files if specified.

Returns

Integer.

Example

```
export_curves -plot Plot_1 -filename testFile.csv -format csv
#-> 0
```

export_movie

Creates a new movie by exporting the selected frames into a GIF file.

NOTE You must use the `start_movie` and `add_frame` commands before using the `export_movie` command.

Syntax

```
export_movie -filename <fileName>
   [-frame_duration <intValue>]
   [-frames {<listOfFrames>}] [-overwrite]
```

Argument	Description
`-filename <fileName>`	Name of the file for the new movie. Add `.gif` extension if necessary.
`-frame_duration <intValue>`	Specifies the duration of each frame with 1/100 s as unit. Default is 50.
`-frames {<listOfFrames>}`	Specifies the list of frames to be exported. Entire frame buffer is used by default.
`-overwrite`	Overwrites existing files if specified.

Returns

String.

Example

```
export_movie -filename Movie.gif -frame_duration 2 -overwrite
#-> Movie.gif
```

See Also

add_frame on page 148
start_movie on page 298

export_settings

Exports Sentaurus Visual settings to a file.

Syntax

```
export_settings <fileName>
```

Argument	Description
`<fileName>`	Name of the file. It must have the file extension `.conf`.

Returns

Integer.

Example

```
export_settings settings.conf
#-> 0
```

export_variables

Exports variables from a curve to a file.

Syntax

```
export_variables -dataset <dataName> -filename <fileName>
   [-overwrite] [<varList>]
```

Argument	Description
`-dataset <dataName>`	Name of dataset where specified (by default, all) variables will be read for export.
`-filename <fileName>`	Specifies the path of the exported file.
`-overwrite`	Overwrites existing files if specified.
`<varList>`	List of variables to be saved. If not specified, the command exports all variables from the dataset provided.

Returns

Integer.

Example

```
export_variables -dataset Data_1 -filename exportedVars.csv
#-> 0
```

export_view

Exports a plot to the specified file format.

If `-plots` is used, the command exports only the specified plots. If it is not specified, the command exports all plots.

Syntax

```
export_view <fileName>
   [-format <stringValue>] [-overwrite] [-plots {<plotList>}]
   [-resolution <width>x<height>]
```

Argument	Description
`<fileName>`	Name of the exported file or files.
`-format <stringValue>`	Specifies the type of file format to use when exporting the plots. The supported formats are BMP, EPS, JPEG, JPG, PNG, PPM, TIF, TIFF, XBM, and XPM.
`-overwrite`	Overwrites existing files if specified.
`-plots {<plotList>}`	Exports the plots specified. If not specified, the command exports all the plots.
`-resolution <width>x<height>`	Specifies the output resolution in pixels.

Returns

Integer.

Example

```
export_view /path/to/examplePlot.bmp -format bmp
#-> 0
```

extract_path

Extracts the path of the maximum or minimum values of a specified scalar field along the horizontal axis. This command returns the name of a new geometry that is added automatically to the 2D plot.

NOTE This command applies only to 2D plots.

Syntax

```
extract_path <fieldName> (-max | -min)
   [-plot <plotName>] [-geom <geometryName>]
   [-materials <stringList> | -regions <stringList>]
   [-window {<x1> <y1> <x2> <y2>}]
```

Argument	Description
`<fieldName>`	Name of the scalar field whose values will be extracted along a path.
`-max | -min`	Specifies whether the command extracts the maximum or minimum values of the scalar field.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.
`-geom <geometryName>`	Specifies the dataset (or geometry) where the command will search for the scalar field. If not specified, the command uses the main one from the active plot.
`-materials <stringList> | -regions <stringList>`	Specifies either a list of materials or a list of regions where the field values will be extracted. If not specified, the command uses the entire 2D plot.
`-window {<x1> <y1> <x2> <y2>}`	Specifies a window defined by `x1`, `y1` (the lower-left corner of the window) and `x2`, `y2` (the upper-right corner of the window). These values must be specified in Cartesian coordinates. If not specified, the command uses the entire 2D plot.

Returns

String.

Example

```
extract_path ElectrostaticPotential -plot Plot_2D -geom 2D
   -materials {Oxide Silicon} -window {-10.32 0 10 10} -max
#-> PathGeometry_2D
```

extract_streamlines

Extracts the fields and coordinates data from one or more streamlines created in 2D or 3D plots.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
extract_streamlines
   {<streamlinesList>} [-plot <plotName>]
```

Argument	Description
{<streamlinesList>}	List of streamlines from which to extract data.
-plot <plotName>	Name of the plot. If not specified, the command uses the selected plot.

Returns

List (names of the 1D datasets created).

Example

```
extract_streamlines {Streamline} -plot Plot_2D
#-> {Streamline(Plot_2D)}
```

```
extract_streamlines {Streamline_1 Streamline_2 Streamline_3} -plot Plot_2D
#-> {Streamline_1(Plot_2D) Streamline_1(Plot_2D) Streamline_1(Plot_2D)}
```

get_axis_prop

Returns axis properties.

NOTE The command returns only one property at a time.

Syntax

```
get_axis_prop -axis (x | y | z | y1 | y2)
(
   -anchor | -auto_padding | -auto_precision | -auto_spacing | -fixed |
   -inverted | -major_ticks_length | -minor_ticks_length |
   -minor_ticks_position | -nof_minor_ticks | -padding | -range |
   -scale_font_att | -scale_font_color | -scale_font_family |
   (-scale_font_size | -scale_font_factor) | -scale_format | -scale_padding |
   -scale_precision | -show | -show_minor_ticks | -show_scale | -show_ticks |
   -show_title | -spacing | -ticks_position | -title | -title_font_att |
   -title_font_color | -title_font_family |
   (-title_font_size | -title_font_factor) | -type
)
   [-plot <plotName>]
```

Argument	Description
`-anchor`	Starting point where ticks are computed.
`-auto_padding`	Shows whether automatic padding of the axis is active (applies to xy plots only).
`-auto_precision`	Returns the property if the precision of the axis is automatic.
`-auto_spacing`	Shows if automatic ticks spacing is used.
`-axis x | y | z | y1 | y2`	Specifies from which axis the properties will be returned. The axis identifiers `y1` and `y2` are only valid for xy plots.
`-fixed`	Shows fixed axis range.
`-inverted`	Shows inverted axis.
`-major_ticks_length`	Length of the major ticks.
`-minor_ticks_length`	Length of the minor ticks.
`-minor_ticks_position`	Shows the position of minor ticks (in, out, or center) (applies to 2D plots only).
`-nof_minor_ticks`	Number of minor ticks on the selected axis.
`-padding`	Padding value of the axis scale, in pixels (applies to xy plots only).

`-plot <plotName>`	Name of the plot from where the axis properties will be returned.
`-range`	Range covered on the axis.
`-scale_font_att`	Font attributes of the axis scale.
`-scale_font_color`	Color of the axis scale.
`-scale_font_family`	Font of the axis scale.
`-scale_font_size` | `-scale_font_factor`	Size (xy plots) or size factor (2D and 3D plots) of the axis scale.
`-scale_format`	Numeric format of the axis scale.
`-scale_padding`	Padding of the axis scale.
`-scale_precision`	Precision of the axis scale.
`-show`	Shows axis.
`-show_minor_ticks`	Shows minor ticks.
`-show_scale`	Shows scale if values for the major ticks are displayed.
`-show_ticks`	Shows major ticks.
`-show_title`	Shows title.
`-spacing`	Shows the spacing between two major ticks.
`-ticks_position`	Shows the position of major ticks (in, out, or center).
`-title`	Axis label.
`-title_font_att`	Font attributes of the axis label.
`-title_font_color`	Color of the axis label.
`-title_font_family`	Font of the axis title.
`-title_font_size` | `-title_font_factor`	Font size (xy plots) or font size factor (2D and 3D plots) of the axis label.
`-type`	Scale type: linear or logarithmic.

Returns

The value of the queried property.

Example

```
get_axis_prop -type
#-> linear
```

get_camera_prop

Returns camera properties.

NOTE The command returns only one property at a time and applies to 3D plots only.

Syntax

```
get_camera_prop [-plot <plotName>]
(
   -focal_point | -position | -rot_color | -rot_size | -rot_width |
   -rotation_point | -show_rotation_point | -zoom
)
```

Argument	Description
`-plot <plotName>`	Name of the plot from which to obtain the required property. If not specified, the command uses the selected plot.
`-focal_point`	Returns position of the focal point.
`-position`	Returns position of the camera.
`-rot_color`	Returns color of the rotation point.
`-rot_size`	Returns size of the rotation point.
`-rot_width`	Returns width of the lines of the rotation point.
`-rotation_point`	Returns position of the rotation point.
`-show_rotation_point`	Shows rotation point.
`-zoom`	Shows the actual zoom of the camera.

Returns

The value of the queried property.

Example

```
get_camera_prop -position
#-> 3.53079 -0.0263978 0.392817
```

get_curve_data

Returns data from the specified curve axis.

Syntax

```
get_curve_data <curveName> (-axisX | -axisY) [-plot <plotName>]
```

Argument	Description
`<curveName>`	Name of the curve from where data is retrieved.
`-axisX | -axisY`	Curve axis from where data is retrieved.
`-plot <plotName>`	Name of plot where curve is displayed.

Returns

List.

Example

```
get_curve_data Curve_1 -axisX
#-> 0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5
```

get_curve_prop

Returns curve properties.

NOTE The command returns only one property at a time and applies to xy plots only.

Syntax

```
get_curve_prop <curveName>
(
   -axis | -color | -deriv | -function | -integ | -label | -line_style |
   -line_width | -markers_size | -markers_type | -show | -show_line |
   -show_markers | -xScale | -yScale | -xShift | -yShift
)
   [-plot <plotName>]
```

Argument	Description
`<curveName>`	Identifier of the curve.
`-axis`	Shows axis alignment of selected curve (left or right aligned).
`-color`	Shows the selected curve line color.
`-deriv`	Shows the order of the derivative applied to the curve.
`-function`	Shows a function applied to the curve.
`-integ`	Shows the integration applied to the curve.
`-label`	Shows the selected curve label.
`-line_style`	Shows the line style of the curve.
`-line_width`	Shows the selected curve line width.
`-markers_size`	Shows the selected curve markers size.
`-markers_type`	Shows the selected curve markers type.
`-show`	Shows curve.
`-show_line`	Shows curve line.
`-show_markers`	Shows activated markers.
`-xScale`	Shows the scale of the curve in the x-axis.
`-yScale`	Shows the scale of the curve in the y-axis.

`-xShift`	Shows the shift of the curve in the x-axis.
`-yShift`	Shows the shift of the curve in the y-axis.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

The value of the queried property.

Example

```
get_curve_prop Curve_1 -deriv
#-> 2
```

get_cutline_prop

Returns the properties of cutlines.

NOTE This command returns only one property at a time. It applies to 2D and 3D plots only.

Syntax

```
get_cutline_prop <cutlineName>
   (-handles_color | -label_op | -label_pos | -label_size | -line_color |
    -line_style | -line_width | -pos1 | -pos2 | -show_handles | -show_label)
   [-plot <stringValue>]
```

Argument	Description
`<cutlineName>`	Name of the cutline from which the property will be returned.
`-handles_color`	Color of the handles.
`-label_op`	The side of the cutline where the label is displayed. It returns false (`0`) if it is the “normal” position or true (`1`) if it is the opposite position.
`-label_pos`	Position of the label.
`-label_size`	Size of the label.
`-line_color`	Color of the line.
`-line_style`	Style of the line.
`-line_width`	Width of the line.
`-pos1`	Position of the first point.
`-pos2`	Position of the second point.
`-show_handles`	The status of the visibility of the handles.
`-show_label`	The status of the visibility of the label.
`-plot <stringValue>`	Name of the plot in which the cutline is located.

Returns

String.

Example

```
get_cutline_prop C1 -plot Plot_2D -pos1
#-> 0.1 2.7891 0
```

get_cutplane_prop

Returns cutplane properties.

NOTE This command applies to 3D plots only.

Syntax

```
get_cutplane_prop <cutplaneName> -plot <plotName>
   (-at | -label_position | -label_size | -normal | -origin | -show_label)
```

Argument	Description
`<cutplaneName>`	Name of the cutplane from which the property will be returned.
`-plot <plotName>`	Name of the plot in which the cutplane is located.
`-at`	Position value of an `-at` type cutplane.
`-label_position`	Position of the label.
`-label_size`	Size of the label.
`-normal`	The normal vector of a free-type cutplane.
`-origin`	The origin vector of a free-type cutplane.
`-show_label`	The status of the visibility of the label.

Returns

String.

Example

```
get_cutplane_prop C1 -plot Plot_3D -at
#-> 0.483
```

get_ellipse_prop

Returns ellipse properties.

NOTE This command returns only one property at a time and applies only to xy plots.

Syntax

```
get_ellipse_prop <objectName>
   (-fill_color | -line_color | -line_style | -line_width | -p1 | -p2)
   [-plot <stringValue>]
```

Argument	Description
`<objectName>`	Name of the ellipse object from which the property will be returned.
`-fill_color`	Returns the color inside of the ellipse.
`-line_color`	Returns the line color.
`-line_style`	Returns the line pattern.
`-line_width`	Returns the line width.
`-p1`	Returns a list of double values representing the start point of the ellipse (upper-left corner).
`-p2`	Returns a list of double values representing the end point of the ellipse (lower-right corner).
`-plot <stringValue>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

The value of the queried property.

Example

```
get_ellipse_prop Ellipse_1 -plot Plot_1 -line_style
#-> dashdotdot
```

get_field_prop

Returns field properties.

NOTE The command returns only one property at a time and applies only to 2D and 3D plots.

Syntax

```
get_field_prop
(
   -custom_levels | -interpolated_values | -label | -levels | -line_color |
   -line_style | -line_width | -max | -max_fixed | -min | -min_fixed |
   -range | -scale | -show | -show_bands
)
   [<fieldName>] [-geom <geometryName>] [-plot <plotName>]
```

Argument	Description
`-custom_levels`	Shows custom levels defined for the selected field.
`-interpolated_values`	Returns whether the interpolated values on its vertices are used for visualization (this property is only valid for fields defined on cells).
`-label`	Label of the field.
`-levels`	Levels of the selected field.
`-line_color`	Color of the contour lines.
`-line_style`	Style of the contour lines.
`-line_width`	Width of the contour lines.
`-max`	Maximum value of the field.
`-max_fixed`	Shows whether the maximum value of the field is fixed.
`-min`	Minimum value of the field.
`-min_fixed`	Shows whether the minimum value of the field is fixed.
`-range`	Range of the selected field.
`-scale`	Scale of the selected field.
`-show`	Shows the selected field on the plot.
`-show_bands`	Shows bands.
`<fieldName>`	Name of the field. If not specified, the command uses the active field.

`-geom <geometryName>`	Name of the dataset (or geometry). If not specified, the command uses the main one from the active plot.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the active plot.

Returns

String.

Example

```
get_field_prop -range
#-> {2.36618e-05 4.36902e+06}
```

get_grid_prop

Returns grid properties of a plot.

NOTE The command applies to xy and 2D plots only.

Syntax

```
get_grid_prop
(
   -align | -line1_color | -line2_color | -line1_style | -line2_style |
   -line1_width | -line2_width | -show | -show_minor_lines
)
   [-plot <plotName>]
```

Argument	Description
`-align`	Shows plot alignment (applies to xy plots only).
`-line1_color`	Shows color of major lines.
`-line2_color`	Shows color of minor lines.
`-line1_style`	Shows style of major lines (applies to xy plots only).
`-line2_style`	Shows style of minor lines (applies to xy plots only).
`-line1_width`	Shows width of major lines.
`-line2_width`	Shows width of minor lines.
`-show`	Shows major lines.
`-show_minor_lines`	Shows minor lines.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

String.

Example

```
get_grid_prop -line1_color
#-> #ff0000
```

get_input_data

Displays a customizable dialog box that requires user input.

NOTE This command works only in interactive mode. Otherwise, it fails unless a default value is provided.

Syntax

```
get_input_data [-default <stringValue>] [-desc <stringValue>]
   [-double | -file | -int | -list | -string | -yesno]
   [-elements {<stringList>}]
```

Argument	Description
`-default <stringValue>`	Sets the default value of a variable.
`-desc <stringValue>`	Text that describes the input.
`-double | -file | -int | -list | -string | -yesno`	Sets the type of input to be expected. Options are: • `-double`: Sets the input to expect a double number. • `-file`: Sets the input to be selected from the file dialog box. • `-int`: Sets the input to expect an integer. • `-list`: Sets the input to be selected from a list. • `-string`: Sets the input to expect a string. This is the default if no type input is specified. • `-yesno`: Sets the input to expect a Yes or No value. It returns 1 for Yes and 0 for No.
`-elements {<stringList>}`	List of possible elements.

Returns

The result of the input as a string.

If you cancel the input, the result is equal to the default value. If a default value is not specified, the command generates an error message.

Example

```
get_input_data -desc "Variable that defines the cut point." -default 0.0 \
   -double
#-> 1.5
```

get_legend_prop

Returns legend properties.

NOTE The command returns only one property at a time.

Syntax

```
get_legend_prop
(
   -color_bg | -color_fg | -label_font_att | -label_font_color |
   -label_font_factor | -label_font_family | -label_font_size |
   -label_format | -location | -margins | -nof_labels | -orientation |
   -position | -precision | -show_background | -size | -title_font_att |
   -title_font_color | -title_font_factor | -title_font_family
)
   [-plot <plotName>]
```

Argument	Description
`-color_bg`	Background color of the legend.
`-color_fg`	Foreground color of the legend.
`-label_font_att`	Font attribute of the legend labels.
`-label_font_color`	Font color of the legend labels.
`-label_font_factor`	Font multiplier of the legend labels.
`-label_font_family`	Font type of the legend labels.
`-label_font_size`	Font size of the legend labels.
`-label_format`	Numeric format of the legend labels.
`-location`	Location in the plot area where the legend is displayed (only for xy plots).
`-margins`	Margins of the legend box.
`-nof_labels`	Number of labels used in the legend.
`-orientation`	Orientation of the legend.
`-position`	Position of the legend that is normalized to the window coordinates between 0 and 1.
`-precision`	Precision of the legend labels.
`-show_background`	Shows whether legend is transparent.
`-size`	Legend size is normalized to window coordinates.

`-title_font_att`	Font attribute of the legend title.
`-title_font_color`	Font color of the legend title.
`-title_font_factor`	Font multiplier of the legend title.
`-title_font_family`	Font type of the legend title.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

String.

Example

```
get_legend_prop -orientation
#-> vertical
```

get_line_prop

Returns line properties.

NOTE This command returns only one property at a time and applies only to xy and 2D plots.

Syntax

```
get_line_prop <objectName>
   (-line_color | -line_style | -line_width | -p1 | -p2)
   [-plot <stringValue>]
```

Argument	Description
`<objectName>`	Name of the line object from which the property is returned.
`-line_color`	Returns the line color.
`-line_style`	Returns the line pattern (applies only to xy plots).
`-line_width`	Returns the line width in pixels.
`-p1`	Returns a list of double values representing the start point of the line.
`-p2`	Returns a list of double values representing the end point of the line.
`-plot <stringValue>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

The value of the queried property.

Example

```
get_line_prop Line_1 -plot Plot_1 -line_width
#-> 4
```

get_material_prop

Returns material properties.

NOTE The command returns only one property at a time and applies only to 2D and 3D plots.

Syntax

```
get_material_prop <materialName>
(
   -border_color | -border_width | -color | -line_color | -on |
   -particles_shape | -particles_size | -show_all | -show_border |
   -show_bulk | -show_field | -show_mesh | -show_vector | -translucency_level |
   -translucency_on | -vector_color
)
   [-geom <geometryName>] [-plot <plotName>]
```

Argument	Description
`<materialName>`	Name of the material.
`-border_color`	Color of material border.
`-border_width`	Width of material border in pixels.
`-color`	Material bulk color.
`-line_color`	Color of line.
`-on`	Status of the visibility of the material.
`-particles_shape`	Shape of the particles.
`-particles_size`	Size of the particles.
`-show_all`	Shows the visibility status of the material.
`-show_border`	Shows the visibility status of the material border.
`-show_bulk`	Shows the visibility status of the material bulk.
`-show_field`	Shows the scalar field visibility status of the material.
`-show_mesh`	Shows the visibility status of the material mesh.
`-show_vector`	Shows the status of the vector field visibility of the material.
`-translucency_level`	Translucency level of the material.
`-translucency_on`	Shows whether translucency is activated.
`-vector_color`	Color of the vector representation inside the material.

`-geom <geometryName>`	Specifies the geometry from where the material property will be requested. If not specified, the command uses first geometry of given `-plot`.
`-plot <plotName>`	Name of the plot from where the material property will be requested. If not specified, the command uses selected plot.

Returns

String.

Example

```
get_material_prop Silicon -plot Plot_2D -show_mesh
#-> false
```

get_plot_prop

Returns plot properties.

NOTE The command returns only one property at a time.

Syntax

```
get_plot_prop
(
   -axes_interchanged | -color_bg | -color_fg | -color_map | -frame_width |
   -keep_aspect_ratio | -ratio_xtoy | -show | -show_axes | -show_axes_scale |
   -show_axes_title | -show_cube_axes | -show_curve_lines |
   -show_curve_markers | -show_grid | -show_legend | -show_major_ticks |
   -show_max_marker | -show_min_marker | -show_minor_ticks | -show_title |
   -tdr_state | -tdr_state_index | -title | -title_font_att |
   -title_font_color | -title_font_factor | -title_font_family |
   -title_font_size | -transformation
)
   [-plot <plotName>]
```

Argument	Description
`-axes_interchanged`	Returns a value indicating whether the axes are interchanged (only for 2D plots).
`-color_bg`	Shows the background color used.
`-color_fg`	Shows the foreground color used.
`-color_map`	Returns a value indicating whether the color map is in default mode (full palette) or grayscale mode.
`-frame_width`	Returns a value that is a positive integer less than 8, denoting the frame width in pixels.
`-keep_aspect_ratio`	Returns a value indicating whether the aspect ratio is maintained (only for 2D plots).
`-ratio_xtoy`	Returns the transformation ratio between the x-axis and y-axis (only for 2D plots).
`-show`	Returns a value indicating whether the plot is displayed.
`-show_axes`	Returns a value indicating whether the axes are present.
`-show_axes_scale`	Returns a value indicating whether axes scales are present (only for xy plots).
`-show_axes_title`	Returns a value indicating whether the axes labels are displayed (only for xy plots).

`-show_cube_axes`	Returns a value indicating whether cube axes are displayed (only for 3D plots).
`-show_curve_lines`	Returns a value indicating whether the curve lines are displayed (only for xy plots).
`-show_curve_markers`	Returns a value indicating whether the markers are enabled (only for xy plots).
`-show_grid`	Returns a value indicating whether the grid is displayed (only for xy and 2D plots).
`-show_legend`	Returns a value indicating whether legend is displayed.
`-show_major_ticks`	Returns a value indicating whether the major ticks are displayed (only for 3D plots).
`-show_max_marker`	Returns a value indicating whether the maximum marker is displayed (only for 2D and 3D plots).
`-show_min_marker`	Returns a value indicating whether the minimum marker is displayed (only for 2D and 3D plots).
`-show_minor_ticks`	Returns a value indicating whether the minor ticks are displayed (only for 3D plots).
`-show_title`	Returns a value indicating whether title is displayed.
`-tdr_state`	Returns the name of the TDR state currently displayed in the plot.
`-tdr_state_index`	Returns the index of the TDR state currently displayed in the plot.
`-title`	Shows the plot label.
`-title_font_att`	Shows the plot label attribute (normal, bold, italic, underline or strikeout).
`-title_font_color`	Shows the plot label color used.
`-title_font_factor`	Returns a value indicating the font factor size (only for 2D and 3D plots).
`-title_font_family`	Shows the label font family used (arial, courier or times).
`-title_font_size`	Returns a value indicating the label font size (only for xy plots).
`-transformation`	Shows the transformation used (only for 3D plots).
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

The value of the queried property.

Example

```
get_plot_prop -title
#-> Test Plot
```

get_rectangle_prop

Returns rectangle properties.

NOTE This command returns only one property at a time and applies only to xy plots and 2D plots.

Syntax

```
get_rectangle_prop <objectName>
   (-fill_color | -line_color | -line_style | -line_width | -p1 | -p2)
   [-plot <stringValue>]
```

Argument	Description
`<objectName>`	Name of the rectangle object from which the property is returned.
`-fill_color`	Returns the color inside of the rectangle.
`-line_color`	Returns the line color.
`-line_style`	Returns the line pattern.
`-line_width`	Returns the line width.
`-p1`	Returns a list of double values representing the start point of the rectangle (lower-left corner in 2D plots or upper-left corner in xy plots).
`-p2`	Returns a list of double values representing the end point of the rectangle (upper-right corner in 2D plots or lower-right corner in xy plots).
`-plot <stringValue>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

The value of the queried property.

Example

```
get_rectangle_prop Rectangle_1 -plot Plot_1 -fill_color
#-> #00FF3B
```

get_region_prop

Returns region properties.

NOTE The command returns only one property at a time and applies only to 2D and 3D plots.

Syntax

```
get_region_prop <regionName>
(
   -border_color | - border_width | -color | -on | -particles_shape |
   -particles_size | -show_all | -show_border | -show_bulk | -show_field |
   -show_mesh | -show_vector | -translucency_level | -translucency_on
)
   [-geom <geometryName>] [-plot <plotName>]
```

Argument	Description
`<regionName>`	Name of the region.
`-border_color`	Color of region border.
`-border_width`	Width of region border in pixels.
`-color`	Region bulk color.
`-on`	Status of the visibility of the material.
`-particles_shape`	Current particles shape. Applies only to particle (kinetic Monte Carlo) regions.
`-particles_size`	Current particles size. Applies only to particle (kinetic Monte Carlo) regions.
`-show_all`	Shows the visibility status of the region.
`-show_border`	Shows the visibility status of the region border.
`-show_bulk`	Shows the visibility status of the region bulk.
`-show_field`	Shows the status of the scalar field visibility of the region.
`-show_mesh`	Shows the visibility status of the region mesh.
`-show_vector`	Shows the status of the vector field visibility of the region.
`-translucency_level`	Translucency level of the region.
`-translucency_on`	Shows whether translucency is activated.

`-geom <geometryName>`	Specifies the geometry from where the region property will be requested. If not specified, the command uses the first geometry of the given `-plot`.
`-plot <plotName>`	Name of the plot from where the region property will be requested. If not specified, the command uses the selected plot.

Returns

String.

Example

```
get_region_prop RGate -plot Plot_2D -border_color
#->#0000FF
```

get_ruler_prop

Returns ruler properties.

NOTE This command returns only one property at a time.

Syntax

```
get_ruler_prop (-color | -precision | -snap_on | -width)
   [-plot <stringValue>]
```

Argument	Description
`-color`	Color of the ruler.
`-plot <stringValue>`	Name of the plot where the command will search for properties. If not specified, the command uses the selected plot.
`-precision`	Decimal precision of the measurements.
`-snap_on`	Status of the snap-to-mesh feature.
`-width`	Width of the ruler in pixels.

Returns

The value of the queried property.

Example

```
get_ruler_prop -width
#-> 4
```

get_streamline_prop

Returns the specified property of a streamline.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
get_streamline_prop <streamlineName>
(
   -arrow_angle | -arrow_color | -arrow_size | -arrow_step | -arrow_width |
   -line_color | -line_resolution | -line_style | -line_width |
   -negative_direction | -show_arrows | -show_line
)
   [-plot <plotName>]
```

Argument	Description
`<streamlineName>`	Name of the streamline from which the specified property will be returned.
`-arrow_angle`	Arrowhead angle.
`-arrow_color`	Color of the arrows.
`-arrow_size`	Size of the arrows.
`-arrow_step`	Step between arrows.
`-arrow_width`	Width of the arrowheads.
`-line_color`	Color of the line.
`-line_resolution`	Distance between the points that conform the line.
`-line_style`	Line style.
`-line_width`	Width of the line.
`-negative_direction`	Shows whether the arrow is displayed in the normal view or inverted view.
`-show_arrows`	Shows whether arrows are visible.
`-show_line`	Shows whether the line is visible.
`-plot <plotName>`	Name of the plot from where the streamline property will be requested. If not specified, the command uses the selected plot.

Returns

String.

Example

```
get_streamline_prop Streamline_1 -plot Plot_2D -arrow_angle
#-> 30
```

get_textbox_prop

Returns the specified property of a text box.

NOTE This command applies to xy and 2D plots only.

Syntax

```
get_textbox_prop <textBoxId>
(
   -anchor_pos | -arrow_size | -font_att | -font_color | -font_factor |
   -font_family | -font_size | -line_color | -line_style | -line_width | -pos |
   -rotation | -show_anchor | -show_border | -text
)
   [-plot <plotName>]
```

Argument	Description
`<textBoxId>`	Name of the text box.
`-anchor_pos`	Returns anchor position using the world coordinate system (only for 2D plots).
`-arrow_size`	Returns arrow size (only for 2D plots).
`-font_att`	Returns font attribute of text box.
`-font_color`	Returns color of text font.
`-font_factor`	Returns multiplier for text font (only for 2D plots).
`-font_family`	Returns font family of text.
`-font_size`	Returns font size (only for xy plots).
`-line_color`	Returns line and arrow color (only for 2D plots).
`-line_style`	Returns the representation style of the text box line (only for 2D plots).
`-line_width`	Returns width of text box and anchor line (only for 2D plots).
`-plot <plotName>`	Name of the plot where the command will search for the text box. If not specified, the command uses the selected plot.
`-pos`	Returns the lower-left corner position of the text box. For xy plots, it returns a point in the world coordinate system {x, y} (see Inserting Text on page 24 for a description of the world coordinate system). For 2D plots, it returns a relative normalized screen coordinates pair (from 0.0 to 1.0).
`-rotation`	Returns the rotation of the text box in degrees (only for xy plots).

`-show_anchor`	Returns true if the text box anchor is shown (only for 2D plots).
`-show_border`	Returns true if the text box border is shown (only for 2D plots).
`-text`	Returns text in text box.

Returns

String.

Example

```
get_textbox_prop Textbox_1 -text
#-> Hello World
```

get_variable_data

Returns a list of variable values.

NOTE This command applies to xy plots only.

Syntax

```
get_variable_data <varName> -dataset <dataName>
```

Argument	Description
`<varName>`	Name of the variable.
`-dataset <dataName>`	Name of the dataset.

Returns

List.

Example

```
get_variable_data X -dataset plotXY
#-> 1 2 3 4 5
```

get_vector_prop

Returns the specified property of a vector field.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
get_vector_prop <vectorField>
(
   -constant_heads | -fill_color | -head_angle | -head_shape | -head_size |
   -line_color | -line_pattern | -line_width | -scale | -scale_factor_grid |
   -scale_factor_uniform | -show
)
   [-geom <geometryName>] [-plot <plotName>]
```

Argument	Description
`<vectorField>`	Name of the vector field.
`-constant_heads`	Shows whether the arrows are constant to the plot area regardless of the vector magnitude or proportional (normal) to the vector magnitude.
`-fill_color`	Color of a solid arrowhead.
`-head_angle`	Arrowhead angle.
`-head_shape`	Shape of the arrows.
`-head_size`	Size of the arrows.
`-line_color`	Color of the arrows.
`-line_pattern`	Line pattern of the arrows.
`-line_width`	Width of the arrows.
`-scale`	Scale for displaying the arrows, either uniform size or a grid display.
`-scale_factor_grid`	Grid factor for displaying the arrows.
`-scale_factor_uniform`	Uniform factor for displaying the arrows.
`-show`	Shows whether arrows are visible.
`-geom <geometryName>`	Specifies the geometry where the command will search for the vector.
`-plot <plotName>`	Name of the plot where the command will search for the geometry. If not specified, the command uses the selected plot.

Returns

String.

Example

```
get_vector_prop ElectricField-V -plot Plot_2D -geom 2D -scale
#-> uniform
```

help

Displays information about Sentaurus Visual Tcl commands.

Without any arguments, the command returns a complete list of the available Tcl commands.

Syntax

```
help [<TclCommand>]
```

Argument	Description
`<TclCommand>`	Sentaurus Visual Tcl command.

Returns

String.

Example

```
help import_settings
# import_settings <filename>
```

import_settings

Imports Sentaurus Visual settings from a previously saved configuration file.

Syntax

```
import_settings <fileName>
```

Argument	Description
`<fileName>`	Name of the configuration file.

Returns

Integer.

Example

```
import_settings /path/to/settings.conf
#-> 0
```

integrate_field

Integrates a field over a complete 2D or 3D plot, or in specific regions.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
integrate_field [-field <fieldName>]
   [-plot <plotName> | -dataset <dataName>]
   [-regions {<regionsList>} | -materials {<materialsList>}]
   [-returndomain]
   [-window {<xmin> <ymin> [<zmin>] <xmax> <ymax> [<zmax>]}]
```

Argument	Description
`-field <fileName>`	Specifies the field to integrate. If not specified, uses the current active field.
`-plot <plotName> | -dataset <dataName>`	Name of the plot or dataset. If not specified, uses the selected plot.
`-regions {<regionsList>} | -materials {<materialsList>}`	List of regions or materials to integrate. If not specified, integrates all regions.
`-returndomain`	Sets the function to return the integrated domain value (an integrated volume when used in three dimensions, or an integrated surface when used in two dimensions).
`-window {<xmin> <ymin> [<zmin>] <xmax> <ymax> [<zmax>]}`	Defines a specific window to integrate. **NOTE** When the plot to be integrated is a 2D plot with x- and z-axes, the parameters change to `{<xmin> <zmin> <xmax> <zmax>}`. In the same way, when the plot has y- and z-axes, the parameters change to `{<ymin> <zmin> <ymax> <zmax>}`.

Returns

Double.

Example

```
integrate_field -field Abs(ElectricField) -regions {RGate}
# Dataset: 3D
# Field: Abs(ElectricField)
# ==================================
# Regions of Dimension  3
# ----------------------------------
# 1. RGate (PolySi)
#    Integral:    3.697957e-04 [V*um^2]
```

```
#    Domain:      9.211180e-04 [um^3]
# ----------------------------------
# Total Integral: 3.697957e-04 [V*um^2]
# Total Domain:   9.211180e-04 [um^3]
# ==================================
#-> 0.000369796
```

link_plots

Links plot properties of two or more plots.

Syntax

```
link_plots {<listOfPlotNames>}
   [-id <intValue>]
   [-special {axes_prop axis_x axis_y axis_z blanking curve_prop cuts
    deformation field_prop field_sel grid_prop legend_prop matreg move
    plot_mode plot_prop streamlines}]
   [-unlink]
```

Argument	Description
`{<listOfPlotNames>}`	List of all the plots to be linked.
`-id <intValue>`	Integer identifier for the linked plots.
`-special {<properties>}`	Links only the specified properties:
`axes_prop`	Links axes properties (only for 2D plots).
`axis_x, axis_y, axis_z`	Links properties of the x-, y-, and z-axis, respectively (only for xy plots).
`blanking`	Links blanking operations (only for 2D and 3D plots).
`curve_prop`	Links curve properties (only for xy plots).
`cuts`	Links cuts such as cutlines and cutplanes (only for 2D and 3D plots).
`deformation`	Links deformation operations (only for 2D and 3D plots).
`field_prop`	Links the properties of fields such as the range, scale, and number of levels (only for 2D and 3D plots).
`field_sel`	Links the selection of fields (on and off for contour band and line fields) (only for 2D and 3D plots).
`grid_prop`	Links grid properties (only for xy plots).
`legend_prop`	Links legend properties.
`matreg`	Links material and region properties.
`move`	Links movements such as zoom, pan, and rotation (only for 3D plots).
`plot_mode`	Links the plot mode, such as select, zoom window, and probe.

`plot_prop`	Links plot properties, except the text of the plot title.
`streamlines`	Links operations on streamlines. Deactivated by default (only for 2D and 3D plots).
`-unlink`	Removes linking.

Returns

Integer.

Example

```
link_plots {Plot_1 Plot_2} -id 10
#-> 10
```

list_curves

Returns a list of curve names.

NOTE The command applies to xy plots only.

Syntax

```
list_curves [<pattern>] [-plot <plotName>] [-show | -hide]
```

Argument	Description
`<pattern>`	Specifies the search pattern to use.
`-plot <plotName>`	Searches on a specific plot. If not specified, the command searches on the selected plot.
`-show | -hide`	Shows or hides the results.

Returns

List.

Example

```
list_curves _1
#-> Curve_1 Curve_12
```

list_custom_buttons

Returns a list of custom buttons.

Syntax

```
list_custom_buttons [<stringValue>]
```

Argument	Description
`<stringValue>`	Specifies the search pattern to use. If not specified, the command lists all custom buttons.

Returns

List of all custom buttons and separators that match the pattern. If no search pattern is specified, the command returns all custom buttons.

Example

```
list_custom_buttons Buttons
#-> Buttons1 Custom_Button2 MyButton
```

list_cutlines

Returns a list of cutlines. If no arguments are specified, the command returns all cutlines.

NOTE The command applies to 2D plots only.

Syntax

```
list_cutlines [<pattern>] [-plot <plotName>] [-type (x | y | z | free)]
```

Argument	Description
`<pattern>`	Specifies the search pattern to use.
`-plot <plotName>`	Searches on a specific plot. If not specified, the command searches on the selected plot.
`-type x | y | z | free`	Specifies the type of cutline to search for.

Returns

List.

Example

```
list_cutlines -type y
#-> C1 C2
```

list_cutplanes

Returns a list of cutplanes. If no arguments are specified, the command returns all cutplanes.

NOTE The command applies to 3D plots only.

Syntax

```
list_cutplanes [<pattern>] [-plot <plotName>] [-type (x | y | z | free)]
```

Argument	Description
`<pattern>`	Specifies the search pattern to use.
`-plot <plotName>`	Searches on a specific plot. If not specified, the command searches on the selected plot.
`-type x | y | z | free`	Specifies the type of cutplane to search for.

Returns

List.

Example

```
list_cutplanes -type y
#-> C3
```

list_datasets

Returns a list of dataset names according to the given pattern. If no pattern is specified, all datasets are returned.

Syntax

```
list_datasets [-dim <x>] [<pattern>]
```

Argument	Description
`-dim <x>`	Dimension of datasets, where `<x>` can be `1`, `2`, or `3`.
`<pattern>`	Specifies the search pattern to use.

Returns

List.

Example

```
list_datasets -dim 3
#-> 3D_3
```

list_ellipses

Returns a list of names of ellipses.

NOTE This command applies to xy plots only.

Syntax

```
list_ellipses [<pattern>] [-plot <stringValue>]
```

Argument	Description
`<pattern>`	Specifies the search pattern to use. If no search pattern is specified, the names of all ellipses are returned.
`-plot <stringValue>`	Name of a plot to search. If not specified, the command uses the selected plot.

Returns

List of names of ellipses matching the search pattern.

Example

```
list_ellipses -plot Plot_1
#-> Ellipse_1 Ellipse_2
```

list_fields

Returns a list of field names.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
list_fields [<pattern>]
   [-plot <plotName> | -dataset <dataName> | -geom <geometryName>]
   [-show | -hide] [-show_bands | -hide_bands]
```

Argument	Description
`<pattern>`	Specifies the search pattern to use.
`-plot <plotName> | -dataset <dataName> | -geom <geometryName>`	Searches a specific plot, dataset, or geometry.
`-show | -hide`	Shows or hides contour bands.
`-show_bands | -hide_bands`	Searches fields with contour bands shown or hidden.

Returns

List.

Example

```
list_fields ElectricField -plot Plot_3D
#-> Abs(ElectricField) ElectricField-X ElectricField-Y ElectricField-Z
```

list_files

Returns a list of opened files according to the given pattern. If no pattern is specified, all files are returned.

Syntax

```
list_files [-fullpath] [<pattern>]
```

Argument	Description
`-fullpath`	Specifies the full path to where the directory resides that contains the opened files. If not specified, uses the current work directory.
`<pattern>`	Specifies the search pattern to use.

Returns

List.

Example

```
list_files
#-> 2D_file.tdr 3D_file.tdr
```

list_lines

Returns a list of line names.

NOTE This command applies to xy and 2D plots only.

Syntax

```
list_lines [<pattern>] [-plot <stringValue>]
```

Argument	Description
`<pattern>`	Specifies the search pattern to use. If no search pattern is specified, the names of all lines are returned.
`-plot <stringValue>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

List of names of lines matching the search pattern.

Example

```
list_lines -plot Plot_1
#-> Line_1 Line_2
```

list_materials

Returns a list of material names.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
list_materials [<pattern>]
   [-plot <plotName> | -dataset <dataName> | -geom <geometryName>]
   ([-show_all | -hide_all] |
     ([-show_border | -hide_border] [-show_bulk | -hide_bulk]
     [-show_field | -hide_field] [-show_mesh | -hide_mesh])
   )
```

Argument	Description
`<pattern>`	Specifies the search pattern to use.
`-plot <plotName> | -dataset <dataName> | -geom <geometryName>`	Searches a specific plot, dataset, or geometry.
`-show_all | -hide_all`	Shows or hides materials completely.
`-show_border | -hide_border`	Shows or hides borders of materials.
`-show_bulk | -hide_bulk`	Shows or hides bulk of materials.
`-show_field | -hide_field`	Shows or hides fields of materials.
`-show_mesh | -hide_mesh`	Shows or hides mesh of materials.

Returns

List.

Example

```
list_materials -plot Plot_3D
#-> Contact DepletionRegion JunctionLine nitride Oxide PolySi Silicon
```

list_movie_frames

Returns the list of frames in the frame buffer.

Syntax

```
list_movie_frames
```

Returns

List.

Example

```
list_movie_frames
# Frame0001 Frame0002 Frame0003 Frame0004
```

list_plots

Returns a list of plot names according to the given pattern. If no pattern is specified, all plots are returned.

Syntax

```
list_plots [-dim <x>] [-link <x>] [<pattern>] [-selected] [-show | -hide]
```

Argument	Description
`-dim <x>`	Dimension of plots, where `<x>` can be `1`, `2`, or `3`.
`-link <x>`	Returns only linked plots with the ID link equal to `<x>`.
`<pattern>`	Specifies the search pattern to use.
`-selected`	Returns the selected plot.
`-show | -hide`	Specifies whether only visible plots (`-show`) or only hidden plots (`-hide`) will be listed.

Returns

List.

Example

```
list_plots -dim 3
#-> 3D
```

list_rectangles

Returns a list of rectangle names.

NOTE This command applies to xy and 2D plots only.

Syntax

```
list_rectangles [<pattern>] [-plot <stringValue>]
```

Argument	Description
`<pattern>`	Specifies the search pattern to use. If no search pattern is specified, the names of all rectangles are returned.
`-plot <stringValue>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

List of names of rectangles matching the search pattern.

Example

```
list_rectangles -plot Plot_1
#-> Rectangle_1 Rectangle_2 Rectangle_3
```

list_regions

Returns a list of region names.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
list_regions [-material <materialName>] [<pattern>]
   [-plot <plotName> | -dataset <dataName> | -geom <geometryName>]
   ([-show_all | -hide_all] |
     ([-show_border | -hide_border] [-show_bulk | -hide_bulk]
     [-show_field | -hide_field] [-show_mesh | -hide_mesh])
   )
```

Argument	Description
`-material <materialName>`	Returns the regions present in the specified material.
`<pattern>`	Specifies the search pattern to use.

`-plot <plotName> |`
`-dataset <dataName> |`
`-geom <geometryName>`
	Returns the regions of the specified plot, dataset, or geometry. If not specified, the command returns the regions of the selected plot.

`-show_all | -hide_all`	Filters by whether regions are completely shown or hidden.
`-show_border | -hide_border`	Filters by whether materials have their border shown or hidden.
`-show_bulk | -hide_bulk`	Filters by whether materials have their bulk shown or hidden.
`-show_field | -hide_field`	Filters by whether materials have their field shown or hidden.
`-show_mesh | -hide_mesh`	Filters by whether materials have their mesh shown or hidden.

Returns

List.

Example

```
list_regions -dataset 3D
#-> bulk gate drain DepletionRegion JunctionLine source
```

list_streamlines

Returns a list of streamlines created on the plot.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
list_streamlines [<pattern>] [-plot <plotName>]
```

Argument	Description
`<pattern>`	Specifies the search pattern to use.
`-plot <plotName>`	Returns the streamlines of the specified plot. If not specified, the command returns the streamlines of the selected plot.

Returns

List.

Example

```
list_streamlines
#-> Streamline_1 Streamline_2 Streamline_3
```

list_tdr_states

Returns a list of state names for the geometry visualized in a plot.

Syntax

```
list_tdr_states [<pattern>] [-plot <plotName>]
```

Argument	Description
`<pattern>`	Specifies the search pattern that the state name must match. If not specified, all state names are returned.
`-plot <plotName>`	Specifies the name of the plot where the geometry of interest is visualized. If not specified, the command uses the selected plot.

Returns

List.

Example

```
list_tdr_states state_1* -plot Plot_2D
#-> state_1 state_10 state_100
```

list_textboxes

Returns a list of text box names.

NOTE This command applies to xy and 2D plots only.

Syntax

```
list_textboxes [<pattern>] [-plot <stringValue>]
```

Argument	Description
`<pattern>`	Specifies the search pattern to use. If no search pattern is specified, the names of all text boxes are returned.
`-plot <stringValue>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

List of names of text boxes matching the search pattern.

Example

```
list_textboxes -plot Plot_1
#-> TextBox_1 TextBox_2 TextBox_3
```

list_variables

Returns a list of variable names according to the given pattern. If no pattern is specified, all variables are returned.

NOTE The command applies to xy plots only.

Syntax

```
list_variables -dataset <dataName> [<pattern>]
```

Argument	Description
`-dataset <dataName>`	Specifies the dataset to use.
`<pattern>`	Specifies the search pattern to use.

Returns

List.

Example

```
list_variables -dataset testDataset
#-> drainCurrent gateToSourceVoltage time
```

load_file

Loads the specified file, and returns a string with the dataset name associated with the file.

Syntax

```
load_file <fileName> -name <stringValue> [-geoms <integerList>]
```

Argument	Description
`<fileName>`	Name of the file to load.
`-name <stringValue>`	Specifies a custom dataset name. See Object Names: -name Argument on page 143.
`-geoms <integerList>`	Specifies the geometry indices to load.

Returns

String.

Example

```
load_file /pathTo/Structure.tdr -geoms {0 2}
#-> Structure_geometry_0
```

load_file_datasets

Loads datasets from the specified file.

Syntax

```
load_file_datasets <fileName> [-geoms <integerList>]
```

Argument	Description
`<fileName>`	Name of the file.
`-geoms <integerList>`	Specifies the geometry indices to load.

Returns

List.

Example

```
load_file_datasets /pathTo/IdVg.tdr
#-> IdVg
```

load_library

Loads a Sentaurus Visual library. It can load either all libraries located at the default paths, or only libraries located at the given path, or only the library given by the specified path and the library name.

Syntax

```
load_library
   -all | (-path <libraryPath> [<libraryName>])
```

Argument	Description
`-all`	Loads all libraries located at the default paths.
`-path <libraryPath> [<libraryName>]`	Path to where libraries for loading are located. Optionally, you can specify the name of a particular library located at the specified path.

Returns

Integer.

Example

```
load_library -path ~/mySVLibs myLibrary1
#-> 0
```

load_script_file

Loads a Tcl script or Inspect script.

Syntax

```
load_script_file <fileName> [-inspect]
```

Argument	Description
`<fileName>`	Name of the Tcl script to load.
`-inspect`	Forces Sentaurus Visual to execute the script as an Inspect script.

Returns

Integer.

Example

```
load_script_file testScript.tcl
#-> 0
```

move_plot

Moves the selected plot.

Syntax

```
move_plot -position {<x> <y>} [-absolute] [-plot <plotName>]
```

Argument	Description
`-position {<x> <y>}`	Sets the new position of the plot. Arguments of type Double are expected.
`-absolute`	Moves plot to an absolute position if specified.
`-plot <plotName>`	Name of the plot to be moved. The current active plot is used if this option is not specified.

Returns

Integer.

Example

```
move_plot -position {1.5 0.5} -absolute
#-> 0
```

overlay_plots

Overlays two or more plots. Creates a new plot with the specified name.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
overlay_plots {<listOfPlots>} [-datasets <listOfDatasets>] [-name <plotName>]
```

Argument	Description
`{<listOfPlots>}`	List of plots to be overlaid onto the first plot. If only one plot is given, the command overlays the plot onto the list of datasets.
`-datasets <listOfDatasets>`	Overlays the list of datasets to a plot given in `<listOfPlots>`.
`-name <plotName>`	Name of the new plot to be created. If not specified, creates a new plot with a generic name. See Object Names: -name Argument on page 143.

Returns

String.

Example

```
overlay_plots {Plot_3D Plot_3D_2} -name Plot_Overlay
#-> Plot_Overlay
```

probe_curve

Probes a curve using the interpolation that matches the axis (linear if the axis is in normal mode or log if the axis is in log mode).

NOTE The command applies to xy plots only. For 2D and 3D plots, use the command `probe_field` (see probe_field on page 243).

Syntax

```
probe_curve <curveName> (-valueX | -valueY <doubleValue>) [-plot <plotName>]
```

Argument	Description
`<curveName>`	Identifier associated with the curve to be probed.
`-valueX | -valueY <doubleValue>`	Specifies the point to probe on the curve.
`-plot <plotName>`	Name of the plot with the curve to be probed. If not specified, the command uses the selected plot.

Returns

Double.

Example

```
probe_curve idvg_1_des -valueX 0.85
#-> 0.5433e-6
```

probe_field

Probes a point on a plot, and returns the values of the defined field on that point.

NOTE This command applies to 2D and 3D plots only. For xy plots, use the command `probe_curve`.

Syntax

```
probe_field
(
   -coord {<x> <y> [<z>]} |
   -point <intValue> -region <regionName>
)
   [-field <fieldName>] [-plot <plotName> | -geom <geometryName>] [-snap]
```

Argument	Description
`-coord {<x> <y> [<z>]}`	Specifies the point to be probed. In 2D plots, the `z` value must be 0 or must be left undefined.
`-point <intValue>`	Specifies the vertex ID relative to the region set to be probed. Use with the `-region` option.
`-region <regionName>`	Specifies the region where the field will be probed. Use with the `-point` option.
`-field <fieldName>`	Name of the field to probe in the plot. If not specified, probes the active field.
`-plot <plotName> |` `-geom <geometryName>`	Name of the plot or geometry to be probed. If not specified, the command probes the selected plot.
`-snap`	Probes the field at the closest node if specified.

Returns

Double.

Example

```
probe_field -field DopingConcentration -coord {0.2 0.3 -0.2}
#-> -2e18
```

reload_datasets

Reloads all the specified datasets.

Syntax

```
reload_datasets {<listOfDatasets>}
```

Argument	Description
{<listOfDatasets>}	List of datasets to reload.

Returns

Integer.

Example

```
reload_datasets {3D 3D_2}
#-> 0
```

reload_files

Reloads the specified files.

Syntax

```
reload_files {<fileNameList>}
```

Argument	Description
{<fileNameList>}	List of files to be reloaded.

Returns

Integer.

Example

```
reload_files {2D.tdr 3D.tdr}
#-> 0
```

remove_curves

Removes curves from an xy plot.

Syntax

```
remove_curves {<listOfCurves>} [-plot <plotName>]
```

Argument	Description
`{<listOfCurves>}`	List of curve names.
`-plot <plotName>`	Name of the plot from where curves will be removed. If not specified, the command uses the active plot.

Returns

Integer.

Example

```
remove_curves {IdVg_1 IdVg_2}
#-> 0
```

remove_custom_buttons

Removes custom buttons.

Syntax

```
remove_custom_buttons {<listOfButtons>}
```

Argument	Description
{<listOfButtons>}	List of names of custom buttons.

Returns

List of all custom buttons and separators removed.

Example

```
remove_custom_buttons {Buttons1 MyButton}
#-> Buttons1 MyButton
```

remove_cutlines

Removes the specified cutlines.

Syntax

```
remove_cutlines {<listOfCutlines>} [-plot <plotName>]
```

Argument	Description
{<listOfCutlines>}	List of cutline names.
-plot <plotName>	Name of plot from where the cutlines will be removed. If not specified, the command uses the active plot.

Returns

List.

Example

```
remove_cutlines {C1 C2}
#-> C1 C2
```

remove_cutplanes

Removes the specified cutplanes.

Syntax

```
remove_cutplanes {<listOfCutplanes>} [-plot <plotName>]
```

Argument	Description
{<listOfCutplanes>}	List of cutplane names. If not specified, the command uses the active plot.
-plot <plotName>	Name of plot from where the cutplanes will be removed.

Returns

List.

Example

```
remove_cutplanes {C1 C2}
#-> C1 C2
```

remove_datasets

Removes the specified datasets.

Syntax

```
remove_datasets {<listOfDatasets>}
```

Argument	Description
{<listOfDatasets>}	List of dataset names.

Returns

Integer.

Example

```
remove_datasets {dataSet1 dataSet2 dataSet3}
#-> 0
```

remove_ellipses

Removes ellipses from a plot.

NOTE This command applies to xy plots only.

Syntax

```
remove_ellipses {<listOfEllipseNames>} [-plot <stringValue>]
```

Argument	Description
`{<listOfEllipseNames>}`	List of ellipse names.
`-plot <stringValue>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

List of deleted ellipses.

Example

```
remove_ellipses {Ellipse_1 Ellipse_2} -plot Plot_1
#-> Ellipse_1 Ellipse_2
```

remove_lines

Removes lines from a plot.

NOTE This command applies to xy and 2D plots only.

Syntax

```
remove_lines {<listOfLineNames>} [-plot <stringValue>]
```

Argument	Description
`{<listOfLineNames>}`	List of line names.
`-plot <stringValue>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

List of deleted lines.

Example

```
remove_lines Line_1 -plot Plot_n60_des
#-> Line_1
```

remove_plots

Removes the specified plots.

Syntax

```
remove_plots {<listOfPlotNames>}
```

Argument	Description
`{<listOfPlotNames>}`	List of plot names.

Returns

Integer.

Example

```
remove_plots {plotXY plot2D plot3D}
#-> 0
```

remove_rectangles

Removes rectangles from a plot.

NOTE This command applies to xy and 2D plots only.

Syntax

```
remove_rectangles {<listOfRectangleNames>} [-plot <stringValue>]
```

Argument	Description
`{<listOfRectangleNames>}`	List of rectangle names.
`-plot <stringValue>`	Name of the plot. If not specified, the command uses the selected plot.

Returns

List of deleted rectangles.

Example

```
remove_rectangles {Rectangle_1 Rectangle_2} -plot Plot_n60_des
#-> Rectangle_1 Rectangle_2
```

remove_streamlines

Removes the specified streamlines.

Syntax

```
remove_streamlines {<listOfStreamlines>} [-plot <plotName>]
```

Argument	Description
`{<listOfStreamlines>}`	List of streamline names.
`-plot <plotName>`	Name of the plot from where the streamlines will be removed. If not specified, the command uses the selected plot.

Returns

List.

Example

```
remove_streamlines {Streamline_1 Streamline_2}
#-> Streamline_1 Streamline_2
```

remove_textboxes

Removes text boxes from a plot.

NOTE This command applies to xy and 2D plots only.

Syntax

```
remove_textboxes {<listOfTextboxNames>} [-plot <stringValue>]
```

Argument	Description
{<listOfTextboxNames>}	List of names of text boxes.
-plot <stringValue>	Name of the plot. If not specified, the command uses the selected plot.

Returns

List of deleted text boxes.

Example

```
remove_textboxes {TextBox_1 TextBox_2 TextBox_3} -plot Plot_1
#-> TextBox_1 TextBox_2 TextBox_3
```

render_mode

Updates the rendering status when plots are loaded. If rendering is switched off, you must switch it on when plots are finished loading; otherwise, no plots will be displayed. Without arguments, this command displays the current status of rendering.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
render_mode [-on | -off]
```

Argument	Description
`-on | -off`	Switches on or off rendering when plots are loaded.

Returns

Current status of rendering mode.

Example

```
render_mode -on
#-> on
```

reset_settings

Reset Sentaurus Visual settings to their default values.

Syntax

```
reset_settings
```

Returns

None.

Example

```
reset_settings
```

rotate_plot

Rotates a 3D plot by specifying either axes, or angles, or a direction, or a plane. Different axes can be rotated simultaneously. The axis and angles used are explained in Figure 48 on page 70.

NOTE This command applies to 3D plots only.

Syntax

```
rotate_plot
(
   -x <degree> -y <degree> -z <degree> |
   -theta <degree> -psi <degree> -alpha <degree> |
   -direction (up | down | left | right) -angle <degree> |
   -plane (xy | yz | xz)
)
   [-absolute] [-plot <plotName>]
```

Argument	Description
`-x <degree>`	Rotates plot around the x-axis (in degrees).
`-y <degree>`	Rotates plot around the y-axis (in degrees).
`-z <degree>`	Rotates plot around the z-axis (in degrees).
`-theta <degree>`	Rotates plot using theta spherical coordinate (in degrees).
`-psi <degree>`	Rotates plot using psi spherical coordinate (in degrees).
`-alpha <degree`	Rotates plot using alpha spherical coordinate (in degrees).
`-direction up | down | left | right`	Rotates the plot in the specified direction by the angle specified by `-angle`.
`-angle <degree>`	Rotates the plot by the specified angle (in degrees) in the direction defined by `-direction`.
`-plane xy | yz | xz`	Rotates the plot in the specified plane.
`-absolute`	Rotates plot to an absolute position if specified.
`-plot <plotName>`	Name of the plot. If not specified, uses the selected plot.

Returns

Integer.

Example

```
rotate_plot -x 10.5 -y 20
#-> 0
```

```
rotate_plot -plane xz
#-> 0
```

```
rotate_plot -direction up -angle 90
#-> 0
```

save_plot_to_script

Exports the plot properties and curve data of the current plot to a Tcl file.

NOTE This command applies to xy plots only.

Syntax

```
save_plot_to_script <filePath> [-overwrite] [-plot <plotName>]
```

Argument	Description
`<filePath>`	Specifies the path (either absolute or relative) where the resulting Tcl file will be located.
`-overwrite`	Specifies whether the target Tcl file should be overwritten if it already exists.
`-plot <plotName>`	Name of the plot to be exported. If not specified, the command uses the selected plot.

Returns

Integer.

Example

```
save_plot_to_script testFile.tcl -plot Plot_1 -overwrite
#-> 0
```

select_plots

Selects the plots.

Syntax

```
select_plots {<listOfPlotNames>}
```

Argument	Description
{<listOfPlotNames>}	List of plot names to be selected.

Returns

List.

Example

```
select_plots {plot2D anotherPlot2D}
#-> plot2D anotherPlot2D
```

set_axis_prop

Sets axis properties.

If `-axis` is not specified, the properties are set for all axes.

NOTE The command applies to xy and 2D plots only.

Syntax

```
set_axis_prop
   [-anchor <doubleValue>]
   [-auto_padding | -manual_padding]
   [-auto_precision | -manual_precision]
   [-auto_spacing | -manual_spacing]
   [-axis (x | y | y1 | y2)]
   [-inverted | -not_inverted]
   [-major_ticks_length <intValue>]
   [-max] [-max_auto | -max_fixed]
   [-min] [-min_auto | -min_fixed]
   [-minor_ticks_length <intValue>]
   [-minor_ticks_position center | in | out]
   [-nof_minor_ticks <intValue>]
   [-padding <intValue>]
   [-plot <plotName>]
   [-range {<x1> <x2>}] [-reset]
   [-scale_font_att (normal | bold | italic | underline | strikeout)]
   [-scale_font_color <#rrggbb>]
   [-scale_font_family (arial | courier | times)]
   ([-scale_font_size <intValue>] | [-scale_font_factor <doubleValue>])
   [-scale_format (preferred | scientific | engineering | fixed)]
   [-scale_padding <intValue>] [-scale_precision <intValue>]
   [-show | -hide] [-show_minor_ticks | -hide_minor_ticks]
   [-show_scale | -hide_scale] [-show_ticks | -hide_ticks]
   [-show_title | -hide_title] [-spacing <doubleValue>]
   [-ticks_position out | in | center] [-title <stringValue>]
   [-title_font_att (normal | bold | italic | underline | strikeout)]
   [-title_font_color <#rrggbb>]
   [-title_font_family (arial | courier | times)]
   ([-title_font_size <intValue>] | [-title_font_factor <doubleValue>])
   [-type linear | log]
```

Argument	Description
`-anchor <doubleValue>`	Sets the anchor of ticks.
`-auto_padding | -manual_padding`	Specifies either automatic padding or manual padding of the axis (applies to xy plots only).
`-auto_precision | -manual_precision`	Sets the automatic or manual precision of the axis.
`-auto_spacing | -manual_spacing`	Sets the spacing mode of ticks.
`-axis x | y | x1 | y2`	Axis to apply the settings. If not specified, the command applies the attributes to all axes.
`-inverted | -not_inverted`	Inverts the axis values.
`-major_ticks_length <intValue>`	Sets the length of major ticks.
`-max`	Sets the maximum value of the axis.
`-max_auto | -max_fixed`	Sets the maximum of the axis automatically, or fixes the maximum of the axis to a user-defined value (applies to xy plots only). If `-max_fixed` is specified, any change to the data will not update the range.
`-min`	Sets the minimum value of the axis.
`-min_auto | -min_fixed`	Sets the minimum of the axis automatically, or fixes the minimum of the axis to a user-defined value (applies to xy plots only). If `-min_fixed` is specified, any change to the data will not update the range.
`-minor_ticks_length <intValue>`	Sets the length of minor ticks.
`-minor_ticks_position center | in | out`	Sets the position of minor ticks (applies to 2D plots only).
`-nof_minor_ticks <intValue>`	Sets the number of minor ticks.
`-padding <intValue>`	Sets the padding of the axis in pixels (applies to xy plots only). If `-auto_padding` is specified, `-padding` has no effect.
`-plot <plotName>`	Name of the plot. If not specified, the command applies the attributes to the selected plot.
`-range {<x1> <x2>}`	Sets the axis range.
`-reset`	Resets axis parameters to default values (applies to xy plots only).
`-scale_font_att normal | bold | italic | underline | strikeout`	Sets the font attributes of the axis scale.
`-scale_font_color <#rrggbb>`	Sets the axis scale color.

`-scale_font_family arial | courier | times`	Sets the axis scale font.
`-scale_font_size <intValue> | -scale_font_factor <doubleValue>`	Sets the font size (xy plots) or size factor (2D and 3D plots) of axis scale.
`-scale_format preferred | scientific | engineering | fixed`	Sets the axis scale numeric format.
`-scale_padding <intValue>`	Sets the padding of the axis scale values.
`-scale_precision <intValue>`	Sets the numeric precision of the axis scale.
`-show | -hide`	Shows or hides the axis.
`-show_minor_ticks | -hide_minor_ticks`	Shows or hides the minor ticks (applies to xy plots only).
`-show_scale | -hide_scale`	Shows or hides the scale.
`-show_ticks | -hide_ticks`	Shows or hides the major ticks.
`-show_title | -hide_title`	Shows or hides the axis label.
`-spacing <doubleValue>`	Sets the spacing between ticks.
`-ticks_position out | in | center`	Sets the position of the ticks.
`-title <stringValue>`	Sets the axis label.
`-title_font_att normal | bold | italic | underline | strikeout`	Sets font attributes of the axis label.
`-title_font_color <#rrggbb>`	Sets the axis label color.
`-title_font_family arial | courier | times`	Sets the axis label font.
`-title_font_size <intValue> | -title_font_factor <doubleValue>`	Sets the axis label font size (xy plots) or font size factor (2D and 3D plots).
`-type linear | log`	Sets the axis scale (applies to xy plots only).

Returns

String.

Example

```
set_axis_prop -axis y1 -title "Drain Current"
#-> 0
```

set_band_diagram

Creates a band diagram. For more details, see Plotting Band Diagrams on page 53.

Syntax

```
set_band_diagram [{<plotList>}]
```

Argument	Description
{<plotList>}	List of plots created with the cutline function.

Returns

Integer.

Example

```
set_band_diagram Plot_1
#-> 0
```

set_best_look

Automatically configures various plot parameters. For more details, see Best Look Option on page 52.

Syntax

```
set_best_look [{<plotList>}]
```

Argument	Description
{<plotList>}	List of plots to apply best look settings.

Returns

Integer.

Example

```
set_best_look {Plot_1 Plot_2}
#-> 0
```

set_camera_prop

Sets camera properties.

NOTE The command applies to 3D plots only.

Syntax

```
set_camera_prop
   [-focal_point {<x> <y> <z>}]
   [-plot <plotName>]
   [-position {<x> <y> <z>}] [-reset]
   [-rot_color <#rrggbb>] [-rot_size <intValue>] [-rot_width <intValue>]
   [-rotation_point {<x> <y> <z>}]
   [-show_rotation_point | -hide_rotation_point] [-zoom <doubleValue>]
```

Argument	Description
`-focal_point {<x> <y> <z>}`	Sets the focal point of the camera.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.
`-position {<x> <y> <z>}`	Sets the position of the camera.
`-reset`	Resets camera settings to their default values.
`-rot_color <#rrggbb>`	Sets the color of the rotation point.
`-rot_size <intValue>`	Sets the size of the rotation point.
`-rot_width <intValue>`	Sets the width of the rotation point.
`-rotation_point {<x> <y> <z>}`	Sets the rotation point of the structure.
`-show_rotation_point | -hide_rotation_point`	Shows or hides the rotation point on the plot.
`-zoom <doubleValue>`	Sets the zoom of the camera.

Returns

Integer.

Example

```
set_camera_prop -rotation_point {0.2 0.35 -0.25}
#-> 0
```

set_curve_prop

Sets curve properties.

NOTE The command applies to xy plots only.

Syntax

```
set_curve_prop {<listOfCurves>} [-plot <plotName>]
   [-axis (left | right)] [-color <#rrggbb>] [-deriv <intValue> | -integ]
   [-function <functionName>] [-label <curveLabel>]
   [-line_style <stringValue>] [-line_width <intValue>]
   [-markers_size <intValue>] [-markers_type <stringValue>] [-reset]
   [-show | -hide] [-show_legend | -hide_legend] [-show_line | -hide_line]
   [-show_markers | -hide_markers]
   [-xScale <doubleValue>] [-yScale <doubleValue>]
   [-xShift <doubleValue>] [-yShift <doubleValue>]
```

Argument	Description
`{<listOfCurves>}`	List of curves on which to apply the specified properties.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.
`-axis left | right`	Plots values on the left or right y-axis.
`-color <#rrggbb>`	Sets color of the curve.
`-deriv <intValue> | -integ`	Either: • Derives the curve, specifying the order of the derivative (either first order or second order). • Integrates the curve.
`-function <functionName>`	Applies a function to a curve. For details on functions, see Appendix C on page 315.
`-label <curveLabel>`	Sets a label to the curve.
`-line_style <stringValue>`	Sets style of the curve line.
`-line_width <intValue>`	Sets line width of the curve line.
`-markers_size <intValue>`	Sets size of the markers.
`-markers_type <stringValue>`	Sets type of markers of the curve.
`-reset`	Resets curve parameters.
`-show | -hide`	Shows or hides the specified curves.
`-show_legend | -hide_legend`	Shows or hides the curve title from the legend.

`-show_line` | `-hide_line`	Shows or hides the curve line.
`-show_markers` | `-hide_markers`	Shows or hides the curve markers.
`-xScale <doubleValue>`	Sets the x-scale of the curve.
`-yScale <doubleValue>`	Sets the y-scale of the curve.
`-xShift <doubleValue>`	Sets the x-shift of the curve.
`-yShift <doubleValue>`	Sets the y-shift of the curve.

Returns

None.

Example

```
set_curve_prop Curve_1 -label "NetActive Field (Cut from structure_1 at x=0)"
```

set_cutline_prop

Changes cutline properties.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
set_cutline_prop <cutlineName> [-plot <stringValue>]
   [-handles_color <#rrggbb>] [-label_normal | -label_op] [-label_pos (0 | 1)]
   [-label_size <doubleValue>] [-line_color <#rrggbb>]
   [-line_style <stringValue>] [-line_width <intValue>]
   [-pos1 {<x> <y> [<z>]}] [-pos2 {<x> <y> [<z>]}]
   [-show_handles | -hide_handles] [-show_label | -hide_label]
```

Argument	Description
`<cutlineName>`	Name of the cutline from which the property will be returned.
`-plot <stringValue>`	Name of the plot in which the cutline is located.
`-handles_color <#rrggbb>`	Sets the color of the handles.
`-label_normal | -label_op`	Sets the side of the label with respect to the endpoint of the label where the label will be displayed.
`-label_pos (0 | 1)`	Sets the label position. The value can be either `0` or `1` indicating the edge of the cutline.
`-label_size <doubleValue>`	Sets the label size of the cutline.
`-line_color <#rrggbb>`	Sets the color of the cutline.
`-line_style <stringValue>`	Sets the style of the cutline.
`-line_width <intValue>`	Sets the width of the cutline.
`-pos1 {<x> <y> [<z>]}`	Sets the position of the first point of the cutline.
`-pos2 {<x> <y> [<z>]}`	Sets the position of the second point of the cutline.
`-show_handles | -hide_handles`	Shows or hides the handles of the cutline.
`-show_label | -hide_label`	Shows or hides the label of the cutline.

Returns

Integer.

Example

```
set_cutline_prop C1 -plot Plot_2D -pos1 {0.1 2.7891 0}
#-> 0
```

set_cutplane_prop

Changes cutplane properties.

NOTE This command applies to 3D plots only.

Syntax

```
set_cutplane_prop <cutplaneName> -plot <plotName>
(
   [-at <atValue>] | ([-origin {<X> <Y> <Z>}] [-normal {<X> <Y> <Z>}])
)
   [-label_position <intValue>] [-label_size <size>]
   [-show_label | -hide-label]
```

Argument	Description
`<cutplaneName>`	Name of the cutplane in which the properties will be changed.
`-plot <plotName>`	Name of the plot in which the cutplane is located.

`-at <atValue> |`
`(-origin {<X> <Y> <Z>}`
` -normal {<X> <Y> <Z>})`
	Sets the values of the position of the cutplane. If the cutplane is an `-at` type, the `-at` property will be available. Otherwise, the origin and normal properties can be changed.

`-label_position <intValue>`	Sets the label position. The value can be either 0, 1, or 2, indicating different corners of the cutplane.
`-label_size <size>`	Sets the label size of the cutplane.
`-show_label | -hide-label`	Shows or hides the label of the cutplane.

Returns

Integer.

Example

```
set_cutplane_prop C1 -plot Plot_3D -at 0.483
#-> 0
```

set_deformation

Sets the deformation properties for a structure, or creates a plot with an already deformed structure.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
set_deformation <vectorField> [-plot <plotName>]
   [-factor <doubleValue> | -reset] [-new_plot]
```

Argument	Description
`<vectorField>`	Name of the vector field to be used to deform the structure.
`-plot <plotName>`	Name of the plot to be used to deform the structure. If not specified, the command uses the selected plot.
`-factor <doubleValue>`	Factor to be applied to the deformation. The value must be greater than zero. If not specified, the default value is 1.
`-reset`	Resets the current deformation.
`-new_plot`	Creates a new plot.

Returns

String (name of affected plot).

Example

```
set_deformation -plot Plot_n60_des Displacement-V -factor 1e2
#-> Plot_n60_des
```

set_ellipse_prop

Sets the properties of an ellipse.

Syntax

```
set_ellipse_prop <stringValue>
   [-fill_color <#rrggbb>] [-line_color <#rrggbb>]
   [-line_style dash | dashdot | dashdotdot | dot | solid]
   [-line_width <intValue>] [-p1 <doubleList>] [-p2 <doubleList>]
   [-plot <stringValue>]
```

Argument	Description
`<stringValue>`	Name of the ellipse.
`-fill_color <#rrggbb>`	Specifies the fill color for the ellipse. Default: `transparent`.
`-line_color <#rrggbb>`	Specifies line color of the ellipse.
`-line_style dash | dashdot | dashdotdot | dot | solid`	Specifies line style of the ellipse.
`-line_width <intValue>`	Specifies the line width.
`-p1 <doubleList>`	Specifies the upper-left corner of the *invisible* rectangle in which the ellipse is drawn.
`-p2 <doubleList>`	Specifies the lower-right corner of the *invisible* rectangle in which the ellipse is drawn.
`-plot <stringValue>`	Name of the plot where the command will search for the ellipse. If not specified, the command uses the selected plot.

Returns

String.

Example

```
set_ellipse_prop Ellipse_1 -fill_color red -line_style dash
#-> 0
```

set_field_prop

Sets field properties.

If `<fieldName>` is not specified, the properties are set for the selected field. The same applies to `-plot`.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
set_field_prop <fieldName>
(
   [-custom_levels {<values>}] |
   ([-scale (linear | log | asinh)] [-levels <intValue>])
)
   [-geom <geometryName>] [-interpolated_values | -primary_values]
   [-label <fieldLabel>] [-line_color <color>] [-line_style <style>]
   [-line_width <width>] [-max] [-max_auto | -max_fixed] [-min]
   [-min_auto | -min_fixed] [-plot <plotName>] [-range {<min> <max>} | -reset]
   [-show | -hide] [-show_bands | -hide_bands]
```

Argument	Description
`<fieldName>`	Name of the field.
`-custom_levels {<values>}`	Specifies a custom list of levels.
`-geom <geometryName>`	Name of the dataset (or geometry). If not specified, the command uses the main one from the active plot.
`-interpolated_values` | `-primary_values`	Specifies whether the primary values of a cell or the interpolated values on its vertices are used for visualization (this property is only valid for fields defined on cells).
`-label <fieldLabel>`	Specifies the label of the selected field.
`-line_color <color>`	Sets the color of the contour lines.
`-line_style <style>`	Sets the style of the contour lines.
`-line_width <width>`	Sets the width of the contour lines.
`-max`	Sets the maximum value of the field.
`-max_auto` | `-max_fixed`	Sets the maximum value of the field automatically, or fixes the maximum value of the field. If `-max_fixed` is specified and, for example, plots are linked, the change to the field values will not update the range.
`-min`	Sets the minimum value of the field.

`-min_auto` | `-min_fixed`	Sets the minimum value of the field automatically, or fixes the minimum value of the field. If `-min_fixed` is specified and, for example, plots are linked, the change to the field values will not update the range.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.
`-range {<min> <max>}` | `-reset`	Sets range of the field contour plot, or resets to the default values.
`-scale (linear | log | asinh)` `-levels <intValue>`	Sets a scale with `<intValue>` levels. Do not use with the `-custom_levels` option.
`-show` | `-hide`	Shows or hides the contour plot.
`-show_bands` | `-hide_bands`	Shows or hides contour bands.

Returns

Integer.

Example

```
set_field_prop -range {-1e20 1e20}
#-> 0
```

set_grid_prop

Sets grid properties.

NOTE The command applies to xy and 2D plots only.

Syntax

```
set_grid_prop [-plot <plotName>]
   [-align (left | right)]
   [-line1_color <#rrggbb>] [-line2_color <#rrggbb>]
   [-line1_style <stringValue>] [-line2_style <stringValue>]
   [-line1_width <intValue>] [-line2_width <intValue>]
   [-reset] [-show | -hide] [-show_minor_lines | -hide_minor_lines]
```

Argument	Description
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.
`-align left | right`	Aligns of the grid to the left or right (applies to xy plots only).
`-line1_color <#rrggbb>`	Sets color of the major grid lines.
`-line2_color <#rrggbb>`	Sets color of the minor grid lines.
`-line1_style <stringValue>`	Sets style of the major grid lines (applies to xy plots only).
`-line2_style <stringValue>`	Sets style of the minor grid lines (applies to xy plots only).
`-line1_width <intValue>`	Sets width of the major grid lines.
`-line2_width <intValue>`	Sets width of the minor grid lines.
`-reset`	Resets plot grid properties (applies to xy plots only).
`-show | -hide`	Shows or hides the major grid lines.
`-show_minor_lines | -hide_minor_lines`	Shows or hides the minor grid lines.

Returns

None.

Example

```
set_grid_prop -show_minor_lines
```

set_legend_prop

Sets legend properties.

These properties apply to xy plots only: `-color_bg`, `-color_fg`, `-label_font_size`, `-location`, and `-margins`.

These properties apply to 2D and 3D plots only: `-label_format`, `-nof_labels`, `-orientation`, and `-precision`.

Syntax

```
set_legend_prop [-plot <plotName>]
   [-color_bg <#rrggbb>] [-color_fg <#rrggbb>]
   [-label_font_att <stringValue>] [-label_font_color <#rrggbb>]
   [-label_font_family (arial | courier | times)]
   [-label_font_size <intValue> | -label_font_factor <doubleValue>]
   [-label_format <stringValue>]
   [-location (top_left | top_right | bottom_left | bottom_right)]
   [-margins {<x1> <x2>}] [-nof_labels <intValue>]
   [-orientation (vertical | horizontal)] [-position {<x> <y>}]
   [-precision <intValue>] [-reset]
   [-show_background | -hide_background] [-size {<x> <y>}]
   [-title_font_att <stringValue>] [-title_font_color <#rrggbb>]
   [-title_font_factor <doubleValue>]
   [-title_font_family (arial | courier | times)]
```

Argument	Description
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.
`-color_bg <#rrggbb>`	Sets background color (apply to xy plots only).
`-color_fg <#rrggbb>`	Sets foreground color (apply to xy plots only).
`-label_font_att <stringValue>`	Sets labels font attribute from: `normal | bold | italic | underline | strikeout`
`-label_font_color <#rrggbb>`	Sets font color of labels.
`-label_font_family arial | courier | times`	Sets labels font.
`-label_font_size <intValue> | -label_font_factor <doubleValue>`	Sets labels font size using either an integer (size) or a factor for resizing the font (factor).
`-label_format <stringValue>`	Sets label format (apply to 2D and 3D plots only).

`-location top_left | top_right | bottom_left | bottom_right`	Sets legend location (apply to xy plots only).
`-margins {<x1> <x2>}`	Sets legend margins (apply to xy plots only).
`-nof_labels <intValue>`	Sets number of labels, (apply to 2D and 3D plots only).
`-orientation vertical | horizontal`	Sets legend orientation (apply to 2D and 3D plots only).
`-position {<x> <y>}`	Sets the legend position that is normalized to the window coordinates between 0 and 1.
`-precision <intValue>`	Sets precision of labels (apply to 2D and 3D plots only).
`-reset`	Resets legend properties.
`-show_background | -hide_background`	Sets the legend background as either solid or translucent.
`-size {<x> <y>}`	Sets the legend size normalized to window coordinates.
`-title_font_att <stringValue>`	Sets legend title font attribute from: `normal | bold | italic | underline | strikeout`
`-title_font_color <#rrggbb>`	Sets font color of legend title.
`-title_font_factor <doubleValue>`	Sets legend title font size using a factor to resize the font.
`-title_font_family arial | courier | times`	Sets legend title font.

Returns

None.

Example

```
set_legend_prop -nof_labels 4 -orientation horizontal
```

set_line_prop

Sets the properties of a line.

Syntax

```
set_line_prop <stringValue>
   [-line_color <#rrggbb>]
   [-line_style dash | dashdot | dashdotdot | dot | solid]
   [-line_width <intValue>] [-p1 <doubleList>] [-p2 <doubleList>]
   [-plot <stringValue>]
```

Argument	Description
`<stringValue>`	Name of a line.
`-line_color <#rrggbb>`	Specifies the line color.
`-line_style dash | dashdot | dashdotdot | dot | solid`	Specifies the line style (only for xy plots).
`-line_width <intValue>`	Specifies the line width.
`-p1 <doubleList>`	Specifies the start point of the line.
`-p2 <doubleList>`	Specifies the end point of the line.
`-plot <stringValue>`	Name of the plot where the command will search for the line. If not specified, the command uses the selected plot.

Returns

String.

Example

```
set_line_prop Line_1 -line_style dot -line_width 2
#-> 0
```

set_material_prop

Sets material properties.

If `-plot` is not specified, the properties are set for the selected material.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
set_material_prop [{<listOfMaterials>}] [-plot <plotName>]
   [-geom <geometryName>]
   [-border_color <#rrggbb>] [-border_width <intValue>] [-color <#rrggbb>]
   [-on | -off]
   [-particles_shape (circle | pentagon | point | sphere | square | triangle)]
   [-particles_size <intValue>]
   [-show_all | -hide_all] [-show_border | -hide_border]
   [-show_bulk | -hide_bulk] [-show_field | -hide_field]
   [-show_mesh | -hide_mesh] [-show_vector | -hide_vector]
   [-translucency_level <doubleValue>]
   [-translucency_on | -translucency_off]
```

Argument	Description
`{<listOfMaterials>}`	List of materials on which to apply the specified properties.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.
`-geom <geometryName>`	Name of the dataset (or geometry). If not specified, uses the main one from the active plot.
`-border_color <#rrggbb>`	Specifies the border color of the materials. Color in format RGB is expected (see Colors on page 144).
`-border_width <intValue>`	Specifies the border width (in pixels) of the selected materials. The default width is 1 pixel with the following exceptions: • For depletion regions and overlays, the default width is 2 pixels. • For junction lines, the default width is 3 pixels. • For contacts and interfaces, the default width is 2 pixels for 2D geometries and 3 pixels for 3D geometries.
`-color <#rrggbb>`	Specifies the bulk color of the materials. Color in format RGB is expected (see Colors).
`-on | -off`	Shows or hides the material.

`-particles_shape (circle | pentagon | point | sphere | square | triangle)`	Sets the shape of particles of particle (kinetic Monte Carlo) materials.
`-particles_size <intValue>`	Sets the size of particles of particle (kinetic Monte Carlo) material.
`-show_all | -hide_all`	Shows or hides all the properties of the materials.
`-show_border | -hide_border`	Shows or hides the border of the materials.
`-show_bulk | -hide_bulk`	Shows or hides the bulk of the materials.
`-show_field | -hide_field`	Shows or hides the scalar fields of the materials.
`-show_mesh | -hide_mesh`	Shows or hides the mesh of the materials.
`-show_vector | -hide_vector`	Shows or hides the vector fields of the materials.
`-translucency_level <doubleValue>`	Specifies the level of translucency when the option `-translucency_on` is specified.
`-translucency_on | -translucency_off`	Activates or deactivates the translucency of the materials.

Returns

Integer.

Example

```
set_material_prop {Oxide Silicon} -show_field
#-> 0
```

set_plot_prop

Sets plot properties.

Syntax

```
set_plot_prop [-plot <plotName>]
   [-axes_interchanged | -not_axes_interchanged]
   [-color_bg <#rrggbb>] [-color_fg <#rrggbb>]
   [-color_map (grayscale | default)] [-contacts_color (constant | list | map)]
   [-frame_width <intValue>] [-keep_aspect_ratio | -not_keep_aspect_ratio]
   [-ratio_xtoy <doubleValue>] [-reset]
   [-show | -hide] [-show_axes | -hide_axes]
   [-show_axes_scale | -hide_axes_scale]
   [-show_axes_title | -hide_axes_title]
   [-show_cube_axes | -hide_cube_axes]
   [-show_curve_lines | -hide_curve_lines]
   [-show_curve_markers | -hide_curve_markers]
   [-show_grid | -hide_grid] [-show_legend | -hide_legend]
   [-show_major_ticks | -hide_major_ticks]
   [-show_max_marker | -hide_max_marker] [-show_min_marker | -hide_min_marker]
   [-show_minor_ticks | -hide_minor ticks] [-show_title | -hide_title]
   [-tdr_state <stringValue> | -tdr_state_index <intValue>]
   [-title <stringValue>] [-title_font_att <stringValue>]
   [-title_font_color <#rrggbb>]
   [-title_font_family (arial | courier | times)]
   [-title_font_size <intValue> | -title_font_factor <doubleValue>]
   [-transformation {<x> <y> <z>}]
```

Argument	Description
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.
`-axes_interchanged` | `-not_axes_interchanged`	Interchanges axes (applies to xy plots only).
`-color_bg <#rrggbb>`	Sets the background color.
`-color_fg <#rrggbb>`	Sets the foreground color.
`-color_map grayscale | default`	Sets the color map used in the plot (applies to 2D and 3D plots only). Values are: • When set to `default`, uses normal color map (full palette). • When set to `grayscale`, uses only grayscale colors.
`-contacts_color constant | list | map`	Sets the behavior of the contact colors. The `list` and `map` arguments must be configured in the user preferences first.

`-frame_width <intValue>`	Sets the plot frame width, which must be a positive integer value less than 8 (applies to xy plots only).
`-keep_aspect_ratio | -not_keep_aspect_ratio`	Configures the aspect ratio (applies to 2D and 3D plots only).
`-ratio_xtoy <doubleValue>`	Sets the x to y ratio of the plot (applies to 2D plots only).
`-reset`	Resets plot properties.
`-show | -hide`	Shows or hides the plot.
`-show_axes | -hide_axes`	Shows or hides the axes.
`-show_axes_scale | -hide_axes_scale`	Shows or hides the axes scale.
`-show_axes_title | -hide_axes_title`	Shows or hides the axes title.
`-show_cube_axes | -hide_cube_axes`	Shows or hides cube axes (applies to 3D plots only).
`-show_curve_lines | -hide_curve_lines`	Shows or hides the curve lines (applies to xy plots only).
`-show_curve_markers | -hide_curve_markers`	Shows or hides the curve markers (applies to xy plots only).
`-show_grid | -hide_grid`	Shows or hides the grid (applies to xy and 2D plots only).
`-show_legend | -hide_legend`	Shows or hides the plot legend.
`-show_major_ticks | -hide_major_ticks`	Shows or hides the major ticks (applies to 3D plots only).
`-show_max_marker | -hide_max_marker`	Shows or hides the maximum marker (applies to 2D and 3D plots only).
`-show_min_marker | -hide_min_marker`	Shows or hides the minimum marker (applies to 2D and 3D plots only).
`-show_minor_ticks | -hide_minor_ticks`	Shows or hides the minor ticks (applies to 3D plots only).
`-show_title | -hide_title`	Shows or hides the plot title.
`-tdr_state <stringValue> | -tdr_state_index <intValue>`	Sets the current TDR state from the state name or the state index (applies to 2D and 3D plots only). To display the last state, specify: `-tdr_state_index -1`
`-title <stringValue>`	Title of the plot.
`-title_font_att <stringValue>`	Sets the title font attribute from: `normal | bold | italic | underline | strikeout`
`-title_font_color <#rrggbb>`	Sets the title font color.
`-title_font_family arial | courier | times`	Sets the title font.

`-title_font_size <intValue> |` `-title_font_factor <doubleValue>`	Sets the title font size (xy plots) or multiplies the font size by a factor (2D and 3D plots).
`-transformation {<x> <y> <z>}`	Sets a linear coordinate transformation (applies to 3D plots only).

Returns

None.

Example

```
set_plot_prop -title "Example 3D Structure"
```

set_rectangle_prop

Sets the properties of a rectangle.

Syntax

```
set_rectangle_prop <stringValue>
   [-fill_color <#rrggbb>] [-line_color <#rrggbb>]
   [-line_style dash | dashdot | dashdotdot | dot | solid]
   [-line_width <intValue>] [-p1 <doubleList>] [-p2 <doubleList>]
   [-plot <stringValue>]
```

Argument	Description
`<stringValue>`	Name of a rectangle.
`-fill_color <#rrggbb>`	Specifies the fill color of the rectangle. Transparency is the default (only for xy plots).
`-line_color <#rrggbb>`	Specifies the line color of the rectangle.
`-line_style dash | dashdot | dashdotdot | dot | solid`	Specifies the line style of the rectangle (only for xy plots).
`-line_width <intValue>`	Specifies the line width of the rectangle.
`-p1 <doubleList>`	Specifies the lower-left corner of the rectangle.
`-p2 <doubleList>`	Specifies the upper-right corner of the rectangle.
`-plot <stringValue>`	Name of the plot where the command will search for the rectangle. If not specified, the command uses the selected plot.

Returns

String.

Example

```
set_rectangle_prop Rectangle_1 -line_width 5
#-> 0
```

set_region_prop

Sets region properties.

If `-plot` is not specified, the properties are set for the selected region.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
set_region_prop [{<listOfRegions>}] [-plot <plotName>] [-geom <geometryName>]
   [-border_color <#rrggbb>] [-border_width <intValue>] [-color <#rrggbb>]
   [-on | -off]
   [-particles_shape (circle | pentagon | point | sphere | square | triangle)]
   [-particles_size <intValue>]
   [-show_all | -hide_all] [-show_border | -hide_border]
   [-show_bulk | -hide_bulk] [-show_field | -hide_field]
   [-show_mesh | -hide_mesh] [-show_vector | -hide_vector]
   [-translucency_level <doubleValue>] [-translucency_on | -translucency_off]
```

Argument	Description
`{<listOfRegions>}`	List of regions of the plot where the properties will be applied.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.
`-geom <geometryName>`	Name of the dataset (or geometry). If not specified, the command uses the main one from the active plot.
`-border_color <#rrggbb>`	Specifies the border color of the region. Color in format RGB is expected (see Colors on page 144).
`-border_width <intValue>`	Specifies the border width (in pixels) of the selected regions. The default width is 1 pixel with the following exceptions: • For depletion regions and overlays, the default width is 2 pixels. • For junction lines, the default width is 3 pixels. • For contacts and interfaces, the default width is 2 pixels for 2D geometries and 3 pixels for 3D geometries.
`-color <#rrggbb>`	Specifies the bulk color of the region. Color in format RGB is expected (see Colors).
`-on | -off`	Shows or hides the region.
`-particles_shape (circle | pentagon | point | sphere | square | triangle)`	Sets the shape of particles of particle (kinetic Monte Carlo) regions.

`-particles_size <intValue>`	Sets the size of particles of particle (kinetic Monte Carlo) regions.
`-show_all` | `-hide_all`	Shows or hides all the properties of the regions.
`-show_border` | `-hide_border`	Shows or hides the border of the regions.
`-show_bulk` | `-hide_bulk`	Shows or hides the bulk of the regions.
`-show_field` | `-hide_field`	Shows or hides the scalar fields of the regions.
`-show_mesh` | `-hide_mesh`	Shows or hides the mesh of the regions.
`-show_vector` | `-hide_vector`	Shows or hides the vector fields of the regions.
`-translucency_level <doubleValue>`	Specifies the level of translucency when the option `-translucency_on` is specified.
`-translucency_on` | `-translucency_off`	Enables or disables translucency of the regions.

Returns

Integer.

Example

```
set_region_prop {source gate drain} -show_mesh
#-> 0
```

set_ruler_prop

Sets ruler properties.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
set_ruler_prop
   [-color <#rrggbb>] [-plot <plotName>] [-precision <intValue>]
   [-snap_on | -snap_off] [-width <intValue>]
```

Argument	Description
`-color <#rrggbb>`	Sets the color of the ruler.
`-plot <plotName>`	Name of the plot where the command will apply the properties. If not specified, the command uses the selected plot.
`-precision <intValue>`	Sets the decimal precision of the measurements.
`-snap_on | -snap_off`	Specifies whether to activate the snap-to-mesh feature.
`-width <intValue>`	Sets the width of the ruler in pixels.

Returns

String.

Example

```
set_ruler_prop -width 5 -precision 2
#-> 0
```

set_streamline_prop

Sets the properties of streamlines.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
set_streamline_prop <streamlineList> [-plot <plotName>]
   [-arrow_angle <intValue>] [-arrow_color <#rrggbb>]
   [-arrow_size <doubleValue>] [-arrow_step <doubleValue>]
   [-arrow_style (solid | dash | dot | dashdot | dashdotdot)]
   [-arrow_width <intValue>] [-constant_arrow | -normal_arrow]
   [-line_color <#rrggbb>] [-line_resolution <doubleValue>]
   [-line_style (solid | dash | dot | dashdot | dashdotdot)]
   [-line_width <intValue>] [-positive_direction | -negative_direction]
   [-show_arrows | -hide_arrows] [-show_line | -hide_line]
```

Argument	Description
`<streamlineList>`	List of the streamlines to be modified.
`-plot <plotName>`	Name of the plot where the command will search for streamlines. If not specified, the command uses the selected plot.
`-arrow_angle <intValue>`	Specifies the arrowhead angle in degrees. It must be between 1 and 89.
`-arrow_color <#rrggbb>`	Specifies the color of the arrows.
`-arrow_size <doubleValue>`	Specifies the size of the arrows.
`-arrow_step <doubleValue>`	Specifies the step between arrows. This value must be greater than the line resolution.
`-arrow_style solid | dash | dot | dashdot | dashdotdot`	Specifies the arrow line style.
`-arrow_width <intValue>`	Specifies the width of the arrowheads.
`-constant_arrow | -normal_arrow`	Specifies whether the size of the arrowheads on screen does not change regardless of the zoom level (`-constant_arrow`), or whether the size of the arrowheads changes on screen depending on the zoom level (`-normal_arrow`).
`-line_color <#rrggbb>`	Specifies the color of the line.
`-line_resolution <doubleValue>`	Specifies the distance between the points that conform the line. Lower values imply better line quality but lower performance.
`-line_style solid | dash | dot | dashdot | dashdotdot`	Specifies the line style.

`-line_width <intValue>`	Specifies the width of the line.
`-positive_direction` | `-negative_direction`	Specifies whether the arrow will be shown in the normal view or inverted view. Default: `-positive_direction`.
`-show_arrows` | `-hide_arrows`	Shows or hides the arrows.
`-show_line` | `-hide_line`	Shows or hides the line.

Returns

Integer.

Example

```
set_streamline_prop Streamline_1 -plot Plot_2D -arrow_angle 45
#-> 0
```

set_tag_prop

Prints text in a box.

The text displayed can be changed only with the argument `-custom_text`. The size of the text depends on the box size.

NOTE The command applies only to 2D and 3D plots.

Syntax

```
set_tag_prop [-plot <plotName>] [-custom_text <displayedText>] [-show | -hide]
```

Argument	Description
`-plot <plotName>`	Names of the plot where the tag will be displayed. If not specified, the command uses the selected plot.
`-custom_text <displayedText>`	Specifies the text to be displayed. The text is not shown unless `-show` is specified.
`-show | -hide`	Shows or hides the text.

Returns

Integer.

Example

```
set_tag_prop -plot Plot_2D -custom_text "Test to be displayed." -show
#->0
```

set_textbox_prop

Sets the specified properties of a text box.

NOTE This command applies to xy and 2D plots only.

Syntax

```
set_textbox_prop <textBoxId> [-plot <plotName>]
   [-anchor_pos {<x> <y>}] [-arrow_size <intValue>]
   [-font_att (normal | bold | italic | underline | strikeout)]
   [-font_color <#rrggbb>] [-font_factor <doubleValue>]
   [-font_family (arial | courier | times)] [-font_size <intValue>]
   [-line_color <#rrggbb>]
   [-line_style (solid | dash | dot | dashdot | dashdotdot)]
   [-line_width <intValue>] [-pos {<x> <y>}] [-rotation <intValue>]
   [-show_anchor | -hide_anchor] [-show_border | -hide_border]
   [-text <stringValue>]
```

Argument	Description
`<textBoxId>`	Name of the text box to be modified.
`-plot <plotName>`	Name of the plot where the command will search for the text box. If not specified, the command uses the selected plot.
`-anchor_pos {<x> <y>}`	Specifies the anchor position using the world coordinate system (only for 2D plots).
`-arrow_size <intValue>`	Specifies the arrow size (only for 2D plots).
`-font_att normal | bold | italic | underline | strikeout`	Specifies the font attribute of the text. Several attributes can be combined using braces, for example: `-font_att {bold italic}`
`-font_color <#rrggbb>`	Specifies the color of the text font.
`-font_factor <doubleValue>`	Specifies the multiplier for the text font (only for 2D plots).
`-font_family arial | courier | times`	Specifies the text font family.
`-font_size <intValue>`	Specifies the font size (only for xy plots).
`-line_color <#rrggbb>`	Specifies the line and arrow color (only for 2D plots).
`-line_style solid | dash | dot | dashdot | dashdotdot`	Specifies the representation style of the text box line (only for 2D plots).
`-line_width <intValue>`	Specifies the width of the text box and anchor line (only for 2D plots).

`-pos {<x> <y>}`	Specifies the lower-left corner position of the text box. For xy plots, this is a point in the world coordinate system {x, y}. For 2D plots, this is a relative normalized screen coordinates pair (from 0.0 to 1.0).
`-rotation <intValue>`	Specifies the rotation of the text box in degrees (only for xy plots).
`-show_anchor | -hide_anchor`	Shows or hides the text box anchor (only for 2D plots).
`-show_border | -hide_border`	Shows or hides the text box border (only for 2D plots).
`-text <stringValue>`	Specifies the text in the text box.

Returns

Integer.

Example

```
set_textbox_prop Textbox_1 -text "Label Text" -font_color #ff0000
#-> 0
```

set_transformation

Applies a transformation to a certain geometry. You can scale a geometry, or shift a geometry, or both scale and shift a geometry.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
set_transformation
(
   -scale {<scaleX> <scaleY> <scaleZ>} |
   -shift {<shiftX> <shiftY> <shiftZ>} |
   (-scale {<scaleX> <scaleY> <scaleZ>} -shift {<shiftX> <shiftY> <shiftZ>})
)
   [-geom <geometryName>] [-plot <plotName>]
```

Argument	Description
`-scale {<scaleX> <scaleY> <scaleZ>}`	Sets or returns the scale value of the axis. The list has the form `{x y z}` where the parameters `x`, `y`, and `z` are positive doubles that represent the scale value applied to each axis.
`-shift {<shiftX> <shiftY> <shiftZ>}`	Sets or returns the shift value of the axis. The list has the form `{x y z}` where the parameters `x`, `y`, and `z` are doubles that represent the shift value applied to each axis.
`-geom <geometryName>`	Name of the geometry in which the transformation will be applied.
`-plot <plotName>`	Name of the plot from which the geometry will be obtained.

Returns

Integer.

Example

```
set_transformation -scale {0.5 1 1}
#-> 0
```

set_value_blanking

Sets value blanking.

If `-field` is not specified, the command uses the selected field.

NOTE The command applies to 2D and 3D plots only.

Syntax

```
set_value_blanking [-field <fieldName>]
   (
     (-less | -greater <doubleValue> [-blank (all | any | inter)]
     [-union | -intersection]) | -reset
   )
   [-cons <intValue>] [-plot <plotName>]
```

Argument	Description
`-less | -greater <doubleValue>`	If you specify `-less`, all values less than the specified value will be blanked. If you specify `-greater`, all values greater than the specified value will be blanked.
`-blank all | any | inter`	Selects the value blanking option where: • `all` is all vertices. • `any` is any vertex. • `inter` is interpolate vertices. If not specified, the command uses the `all` option.
`-cons <intValue>`	Number of the value blanking rule. Options are between 1 and 8. Default: 1.
`-field <fieldName>`	Name of the field to set blanking parameters.
`-plot <plotName>`	Name of the plot. If not specified, the command uses the selected plot.
`-reset`	Removes value blanking rules.
`-union | -intersection`	Sets whether the constraints will be united or will intersect. If not specified, the command uses the `-union` option.

Returns

Integer.

Example

```
set_value_blanking -field DopingConcentration -greater 0.0
#-> 0
```

set_vector_prop

Sets the properties of a vector field.

NOTE This command applies to 2D and 3D plots only.

Syntax

```
set_vector_prop <vectorField> [-plot <plotName>] [-geom <geometryName>]
   [-constant_heads | -normal_heads] [-fill_color <#rrggbb>]
   [-head_angle <intValue>]
   [-head_shape (arrow | arrow_solid | head | head_closed | head_solid)]
   [-head_size <doubleValue>]
   [-line_color <#rrggbb>]
   [-line_pattern (solid | dash | dot | dashdot | dashdotdot)]
   [-line_width <intValue>] [-scale (grid | uniform)]
   [-scale_factor_grid <doubleValue>]
   [-scale_factor_uniform <doubleValue>]
   [-show | -hide]
```

Argument	Description
`<vectorField>`	Name of the vector field to be modified.
`-plot <plotName>`	Name of the plot where the command will search for the geometry. If not specified, the command uses the selected plot.
`-geom <geometryName>`	Specifies the geometry where the command will search for the vector.
`-constant_heads | -normal_heads`	Specifies whether the arrows are constant to the plot area regardless of the vector magnitude or proportional (normal) to the vector magnitude.
`-fill_color <#rrggbb>`	Specifies the color of a solid arrowhead. Otherwise, this argument has no effect.
`-head_angle <intValue>`	Specifies the arrowhead angle in degrees. It must be between 1 and 89.
`-head_shape arrow | arrow_solid | head | head_closed | head_solid`	Specifies the shape of the arrows.
`-head_size <doubleValue>`	Specifies the length of the arrows.
`-line_color <#rrggbb>`	Specifies the color of the arrows.
`-line_pattern solid | dash | dot | dashdot | dashdotdot`	Specifies the line pattern of the arrows.
`-line_width <intValue>`	Specifies the width of the arrows.

`-scale grid | uniform`	Specifies the scale for displaying the arrows, either uniform size or a grid display.
`-scale_factor_grid <doubleValue>`	Specifies the grid factor for displaying the arrows.
`-scale_factor_uniform <doubleValue>`	Specifies the uniform factor for displaying the arrows.
`-show | -hide`	Shows or hides the arrows.

Returns

Integer.

Example

```
set_vector_prop ElectricField-V -plot Plot_2D -geom 2D -scale grid
#-> 0
```

set_window_full

Sets the full plot view.

Syntax

```
set_window_full (-on | -off)
```

Argument	Description
`-on | -off`	Activates or deactivates the full plot view.

Returns

Integer.

Example

```
set_window_full -on
#-> 0
```

set_window_size

Sets the size of the main window of the GUI.

Syntax

```
set_window_size <width>x<height>
```

Argument	Description
`<width>x<height>`	Sets the width and the height of the main window in pixels (minimum of 200x200 pixels).

Returns

Integer.

Example

```
set_window_size 1280x800
#-> 0
```

show_msg

Displays a message in a dialog box.

Syntax

```
show_msg [-error | -info | -warning] [-title <stringValue>] <stringMessage>
```

Argument	Description
`-error | -info | -warning`	Specifies the type of message to display. Default: `-info`.
`-title <stringValue>`	Specifies the title of the dialog box.
`<stringMessage>`	Specifies the text to be displayed.

Returns

None.

Example

```
show_msg -warning -title "Bad Value" "There was a problem extracting the
threshold voltage"
```

start_movie

Starts the recording of a new movie by creating a new frame buffer.

NOTE This command only starts the creation of a movie, so you must use the `add_frame` and `export_movie` commands to complete the operations.

Syntax

```
start_movie [-resolution <width>x<height>]
```

Argument	Description
`-resolution <width>x<height>`	Specifies the resolution of each captured frame in pixels. If not specified, uses the current screen resolution.

Returns

None.

Example

```
start_movie
```

stop_movie

Stops recording a movie.

NOTE This command deletes the stored frame buffer. It does not save it into a file.

Syntax

```
stop_movie
```

Returns

None.

Example

```
stop_movie
```

undo

Undoes the last command implemented or the number of commands specified.

Syntax

```
undo [times]
```

Argument	Description
`times`	Number of commands to be reverted.

Returns

None.

Example

```
undo 2
```

unload_file

Unloads all the datasets belonging to the specified file.

Syntax

```
unload_file <fileName>
```

Argument	Description
`<fileName>`	Name of the file.

Returns

Integer.

Example

```
unload_file structure2D.tdr
#-> 0
```

version

Returns the version of Sentaurus Visual.

Syntax

```
version
```

Returns

None.

Example

```
version
#-> 25.0.7
```

windows_style

Specifies the type of window style to use for the Sentaurus Visual GUI.

Syntax

```
windows_style
   [-aspect_ratio_on | -aspect_ratio_off]
   [-direction (right_down | down_right)]
   [-max <numCols>]
   [-sort {<plotList>}]
   [-style (horizontal | vertical | grid | max | custom)]
```

Argument	Description
`-aspect_ratio_on` | `-aspect_ratio_off`	Specifies whether the aspect ratio is maintained for all the plots displayed.
`-direction right_down` | `down_right`	Specifies the viewing direction of plots and where they will stretch: • When using the `right_down` direction, the grid fills to the right until it is full and then continues adding new plots in a new row downwards from the first row. • When using the `down_right` direction, this order is inverted.
`-max <numCols>`	Specifies the maximum number of columns in which to display the plots when they are in a custom grid configuration.
`-sort {<plotList>}`	Specifies the plots to be displayed.
`-style horizontal` | `vertical` | `grid` | `max` | `custom`	Specifies a horizontal or vertical orientation, or grid style, or the use of maximum space or custom style.

Returns

Integer.

Example

```
windows_style -style grid
#-> 0
```

zoom_plot

Zooms into a plot.

Syntax

```
zoom_plot
(
   -axis (x | y | z) -range {<min> <max>} |
   -box {<minX> <maxX> <minY> <maxY> <minZ> <maxZ>} |
   -factor <doubleValue> |
   -reset |
   -window {<x1> <y1> <x2> <y2> [<yr1> <yr2>]} |
)
   [-plot <plotName>]
```

Argument	Description
`-axis (x | y | z)` `-range {<min> <max>}`	Specifies the axis where the range will be applied. Defines the minimum and maximum values of the range.
`-box {<minX> <maxX> <minY> <maxY> <minZ> <maxZ>}`	Defines the three ranges for the boundary box.
`-factor <doubleValue>`	Sets zoom factor. If the value is greater than 1, it zooms into a plot. If the value is smaller than 1, it zooms out of a plot.
`-reset`	Resets the zoom status of the plot.
`-window {<x1> <y1> <x2> <y2> [<yr1> <yr2>]}`	Sets zoom window. It zooms into the specified window between the two x,y pairs. For 2D and 3D plots, the argument only accepts four values. For 3D plots, the values can be entered in the form of pixels of the plot frame (integers) or can be normalized to screen values (doubles between 0 and 1).
`-plot <plotName>`	Name of the plot.

Returns

None.

Example

```
zoom_plot -reset
```

APPENDIX B Menus and Toolbars of Graphical User Interface

This appendix describes the menus and the toolbars of the graphical user interface of Sentaurus Visual.

Menus

This section lists the commands of the different menus.

File Menu

Table 13 File menu

Command	Button	Shortcut keys	Description
Open		Ctrl+O	Loads a dataset or multiple datasets.
Reload All		F5 key	Reloads all loaded datasets.
Reload Selected		Shift+F5	Reloads only the selected datasets.
Export Plot		Ctrl+E	Exports the selected plots to an image.
Import Image		Ctrl+I	Displays the Import Image dialog box.
Run Tcl Script			Runs Tcl or Inspect scripts.
Print Plots		Ctrl+P	Prints the selected plots.
Recent Files			Lists the recently opened datasets, up to five.
Exit		Ctrl+Q	Quits Sentaurus Visual.

Edit Menu

Table 14 Edit menu

Command	Button	Shortcut keys	Description
Undo		Ctrl+Z	Reverts the last operation executed.
Select All Plots		Ctrl+A	Selects all the active plots.
Redraw All Plots		Ctrl+R	Redraws all the active plots.
Delete Selected Plots		Ctrl+D	Deletes all the selected plots.
Preferences			Displays User Preferences dialog box.
Draw			Displays the Draw toolbar for drawing lines, rectangles, and ellipses, and inserting text. Available for xy and 2D plots only.

View Menu

Table 15 View menu

Command	Button	Shortcut keys	Description
Panes			Shows or hides the Selection panel, Properties panel, and Tcl Command panel.
Toolbars			Shows or hides the File, Edit, View, Tools, and Movies toolbars.
Select			Enables selection (default) mode.
Select/Rotate			Enables selection (default) mode. Available for 3D points only.
Reset		Ctrl+Shift+F	Resets plot to the default values.
Zoom		Ctrl+Shift+Z	Enables zoom tool.
Scale to Image			Displays the Scale to Image dialog box, where you can overlay an image onto a plot.
Zoom to Ranges			Displays the Zoom to Ranges dialog box, where you can zoom by specifying the range of one of the three axes using the **Box** tab. Available for 3D plots only.
Best Look		Ctrl+Shift+L	Adjusts plotting parameters automatically. Available for xy plots only.

Table 15 View menu

Command	Button	Shortcut keys	Description
Spherical Rotation			Performs a spherical rotation of the view. Available for 3D plots only.
Rotation Axis X			Fixes the rotation of a 3D plot to the x-axis. Available for 3D plots only.
Rotation Axis Y			Fixes the rotation of a 3D plot to the y-axis. Available for 3D plots only.
Rotation Axis Z			Fixes the rotation of a 3D plot to the z-axis. Available for 3D plots only.
Rotate			Displays Rotate dialog box for rotate modes and angles for 3D plots. Available for 3D plots only.
View Plane XY			Shows a 3D plot in the xy plane. Available for 3D plots only.
View Plane YZ			Shows a 3D plot in the yz plane. Available for 3D plots only.
View Plane XZ			Shows a 3D plot in the xz plane. Available for 3D plots only.
Default View			Restores a 3D plot point of view. Available for 3D plots only.
Fast Draw			If selected, 3D plot becomes an outline during a rotation or move. Available for 3D plots only.
Subsampling			Enables or disables subsampling in 2D and 3D plots. Available for 2D and 3D plots only.
Camera Configuration			Camera configuration for 3D plots. Available for 3D plots only.
Lights Configuration			Lighting parameters for 3D plots. Available for 3D plots only.

Tools Menu

Table 16 Tools menu

Command	Button	Shortcut keys	Description
Link		Ctrl+L	Links two or more plot properties.
Special Link			Displays Special Link dialog box where you can set up special linking to link only specified properties.
Movies			Provides commands to start recording a movie, to add frames to a movie, and to stop recording a movie.
Probe		Ctrl+Shift+P	Probes the values on a plot.
Analysis			Performs analysis on a curve. Available for xy plots only.
Calculate Scalar			Displays the Calculate Scalar dialog box, where you can create a function to calculate scalar values. Available for xy plots only. See Calculate Scalar Tool on page 55.
Precision Cuts			Displays the Cutlines and Cutplanes dialog box. Available for 2D and 3D plots only.
Cutline		Ctrl+Shift+C	Generates a custom cutline on a 2D plot. Available for 2D and 3D plots only.
Cut X		Ctrl+Shift+X	Generates a cutplane (3D) or cutline (2D) orthogonal to the x-axis. Available for 2D and 3D plots only.
Cut Y		Ctrl+Shift+Y	Generates a cutplane (3D) or cutline (2D) orthogonal to the y-axis. Available for 2D and 3D plots only.
Cut Z		Ctrl+Shift+Z	Generates a cutplane (3D) or cutline (2D) orthogonal to the z-axis. Available for 2D and 3D plots only.
Ruler		Ctrl+Shift+R	Enables measuring distances. Available for 2D and 3D plots only.
Value Blanking		Ctrl+Shift+V	Displays Value Blanking dialog box. Available for 3D plots only.
Streamlines			Displays Streamlines dialog box where you can enable drawing streamlines of a vector field. Available for 2D and 3D plots only.

Table 16 Tools menu

Command	Button	Shortcut keys	Description
Overlay		Ctrl+Shift+Y	Overlays two or more plots onto one plot. Available for 2D and 3D plots only.
Diff Plots			Enables tool to plot the difference between common fields. Available for 2D and 3D plots only.
Integrate	$\int dr$		Displays the Field Integration dialog box, where you can perform integration over the active field of a plot. Available for 2D and 3D plots only.
Create Projection			Displays the 2D Projection dialog box, where you can create a 2D minimum or maximum projection of a field from a 3D plot. Available for 3D plots only.
Deformation			Displays Deformation dialog box, where you can create a deformed structure in the same plot or in a new one. Available for 2D and 3D plots only.
Min/Max Field Value			Displays Minimum/Maximum Field Value dialog box, where you can select certain regions or materials for the search, and you can define a 3D box limiting the search area. Available for 2D and 3D plots only.
Create Isovalue			Displays Create Isovalue Geometry dialog box, where you can create a new geometry from a constant field value in a structure. Available for 2D and 3D plots only.
Surface Plot			Displays Surface Plot dialog box, where you can create a surface plot from a 3D dataset.
Extract Path			Displays Extract Path dialog box, where you can extract a path of either the minimum or maximum values of a specified scalar field.

Data Menu

Table 17 Data menu

Command	Button	Shortcut keys	Description
View Info Loaded			Displays Manage Loaded Data dialog box, showing all the datasets and plots currently loaded.
Curve Properties		Ctrl+Shift+E	Displays Curve Properties dialog box. Available for xy plots only.
Region Properties		Ctrl+Shift+E	Displays Region Properties dialog box. Available for 2D and 3D plots only.
Export XY Data			Displays Export XY Data dialog box. Available for xy plots only.
Save Plot			Displays a dialog box where you can save all the plot data and settings to a Tcl file. Available for xy plots only.
New XY Plot			Generates a new empty xy plot. Available for xy plots only.
Duplicate Plot			Duplicates the current plot as an xy plot. Available for xy plots only.
TDR Tags			Displays TDR Tags dialog box, where you can select which tags to display on the selected plot. Available for 2D and 3D plots only.
Dataset Information			Displays Dataset Information dialog box, where you can access 2D or 3D dataset information, such as the number of points or elements in a specific material or region. Available for 2D and 3D plots only.

Window Menu

Table 18 Window menu

Command	Button	Shortcut keys	Description
Tile Grid			Organizes loaded plots into a grid.
Tile Vertically			Organizes loaded plots vertically in the plot area.
Tile Horizontally			Organizes loaded plots horizontally in the plot area.

Table 18 Window menu

Command	Button	Shortcut keys	Description
Set Default State			Restores toolbars and workspace to their default positions in the GUI.
Manage Frames			Displays the Manage Frames dialog box.
Previous Plot		Page Up key	Moves to the previous loaded plot.
Next Plot		Page Down key	Moves to the next loaded plot.
Minimize Plot			Minimizes the selected plot.
Maximize		F10 key	Maximizes the selected plot.
Full Plot View		F12 key	Hides the toolbars and zooms into a plot using the entire workspace.
Restore All Plots			Restores all minimized plots.
Plots			Lists the open plots.

Help Menu

Table 19 Help menu

Command	Button	Shortcut keys	Description
User Guide			Displays PDF of *Sentaurus™ Visual User Guide*.
Tutorial			Displays Sentaurus Visual module of TCAD Sentaurus Tutorial (HTML).
About		Ctrl+B	Shows information about Sentaurus Visual.

NOTE The default viewer for the PDF of the *Sentaurus™ Visual User Guide* is Adobe® Reader®. Using another PDF viewer might disable some cross references or external links in the PDF files.

Toolbars

File Toolbar

Table 20 File toolbar

Button	Description	Button	Description
	Open		Export Plot
	Reload All		Run Tcl Script
	Reload Selected		Print Plots

Edit Toolbar

Table 21 Edit toolbar

Button	Description	Button	Description
	Undo		Draw (xy and 2D plots only)

Draw Toolbar

Table 22 Draw toolbar (available for xy and 2D plots only)

Button	Description	Button	Description
	Draw Line		Draw Ellipse
	Draw Rectangle		Insert Text

View Toolbar

Table 23 View toolbar

Button	Description	Button	Description
	Select Select/Rotate (3D plots only)		Rotation Axis X (3D plots only)
	Reset		Rotation Axis Y (3D plots only)
	Zoom		Rotation Axis Z (3D plots only)
	Best Look (xy plots only)		View Plane XY (3D plots only)
$\log_X$	Log Scale X (xy plots only)		View Plane YZ (3D plots only)
$\log_Y$	Log Scale Y (xy plots only)		View Plane XZ (3D plots only)
$\log_{Y2}$	Log Scale Y Right (xy plots only)		Fast Draw (3D plots only)
	Spherical Rotation (3D plots only)		

Tools Toolbar

Table 24 Tools toolbar

Button	Description	Button	Description
	Link		Cut X (2D and 3D plots only)
	Special Link		Cut Y (2D and 3D plots only)
	Curve Properties (xy plots only)		Cut Z (2D and 3D plots only)
	Region Properties (2D and 3D plots only)		Ruler (2D and 3D plots only)
	Probe		Value Blanking (3D plots only)
	Analysis (xy plots only)		Streamlines (2D and 3D plots only)

Table 24 Tools toolbar

Button	Description	Button	Description
	Plot Band Diagram (xy plots only)		Overlay (2D and 3D plots only)
	Precision Cuts (2D and 3D plots only)		Diff Plots (2D and 3D plots only)
	Cutline (2D plots only)	$\int dr$	Integrate (2D and 3D plots only)

Movies Toolbar

Table 25 Movies toolbar

Button	Description	Button	Description
	Start Recording		Add Frames
	Stop Recording		

Look Toolbar

Table 26 Look toolbar

Button	Description	Button	Description
	Change Panel View (Changes presentation of left pane from separate tabs to one view)	Data	Selection Panel (Shows or hides Selection panel)
Prop	Properties Panel (Shows or hides properties panel for whichever plot is selected)	Tcl	Tcl Command Panel (Shows or hides Tcl Command panel)

Additional Keyboard Shortcuts (2D and 3D Plots)

Table 27 Additional keyboard shortcuts for 2D and 3D plots

Action	Shortcut keys	Description
Basic rotation	Press the N key while dragging the cursor	Enables *rollerball* rotation until you release the N key (applies to 3D plots only).
Rotate around x-axis	Press the X key while dragging the cursor	Enables rotation around the x-axis until you release the X key (applies to 3D plots only).
Rotate around y-axis	Press the Y key while dragging the cursor	Enables rotation around the y-axis until you release the Y key (applies to 3D plots only).
Rotate around z-axis	Press the Z key while dragging the cursor	Enables rotation around the z-axis until you release the Z key (applies to 3D plots only).
Spherical rotation	Press the S key while dragging the cursor	Enables spherical rotation until you release the S key (applies to 3D plots only).
Change rotation center point	Press O key	Updates the rotation point of the structure. A new point will be placed at the cursor position.
Switch on or off 3D guide axis	Press I key	Switches on or off the 3D guide axis. However, to see this change, a minor rotation is required.
Enable zoom navigation	Press Ctrl+Shift while dragging the cursor	Enables zoom navigation (equivalent to clicking and dragging the middle mouse button).
Zoom to cursor position	Press F key	When you place the cursor somewhere on the structure (you do not click) and then press the F key, the structure changes view so that the new center of the plot is where the cursor was placed.
Highlight material or region menu	Double-click	Highlights the region or material in blue in the Selection panel.
Highlight region	Press P key	Highlights the selected region using a red box. To cancel this operation, press the P key when the cursor is not positioned over any region.
Reset view	Press R key	Returns the structure to the default view.
Enable wireframe view	Press W key	Changes the display of the structure to a mesh view.
Enable solid view	Press S key	Changes the display of the structure to a non-mesh view.

APPENDIX C Available Formulas

This appendix presents an overview of the functions available in Sentaurus Visual as well as the syntax of the formulas used to create curves, variables, and fields.

Creating a New Variable

NOTE For new variables, variables of an existing dataset can be used in the function specification or a list of values.

To create a new variable, use the `create_variable` command. For example, to create the common logarithm of the variable Y present in the dataset `myDataset` as a new variable, you can use the command:

```
create_variable -name commonLogY -dataset "myDataset"
   -function "log(<Y:myDataset>)"
```

To access variables on functions, use the format `<VARIABLE:DATASET>`. This new variable will appear in the variables list of the dataset in which it was created.

NOTE Variables can be created on the **Data** tab of the Selection panel by clicking the **New Variable** button. A dialog box is displayed where you can interactively add functions, operators, and variables to create a new formula.

Creating a New Curve

NOTE For new curves, existing curves can be used in the function specification.

To create a new curve, use the `create_curve` command. For example, to create the derivative of `Curve_1` and name it `newCurve`, you can use the command:

```
create_curve -name newCurve -function diff(<Curve_1>)
```

To use the curves on formulas, you must write the curve identifier in angle brackets. For example, to use the data on `Curve_1` for the differentiation function, it is written as `<Curve_1>`.

NOTE If you want to create a new curve from more than one curve using a function, for example:

```
create_curve -name newCurve_2 -function <Curve_1>*<Curve_2>
```

both curves `Curve_1` and `Curve_2` must share the same x-axis and must have the same amount of valid data. Otherwise, this could lead to unexpected results.

NOTE Curves can be created on the **Curves** tab of the Selection panel by clicking the **New** button. A dialog box is displayed where you can interactively add functions, operators, and curves to create a new curve based on a formula.

Applying Functions to a Curve

To apply a function to an existing curve, use the `set_curve_prop` command. For example, to apply the absolute value function to Curve_1, use the command:

```
set_curve_prop Curve_1 -function "abs"
```

Alternatively, you can use the Curve Properties panel:

1. Select the curve.
2. Click the **Trans.** tab.
3. From the **Function** list, select the required function.

NOTE It is not possible to apply more than one function to an existing curve. Instead, it is recommended to create a new curve.

Furthermore, as an exception, you can apply the integral, or a first-derivative or second-derivative function in addition to the other function, using the same command, but with another parameter (`-integ` or `-deriv`):

```
set_curve_prop Curve_1 -integ
set_curve_prop Curve_1 -deriv 2
```

Alternatively, you can use the Curve Properties panel:

1. Select the curve.
2. Click the **Trans.** tab.
3. From the **Deriv / Integ** list, select the function to apply.

Creating a New Field

NOTE For new fields, existing fields can be used in the function specification.

To create a new field, use the `create_field` command. Existing fields are used to create new fields based on functions and operations specified by the user. In the following example, consider two fields called `ElectricField-X` and `ElectricField-Y`. You want to create a new field that contains the absolute value of the sum of both fields. This can be done with the following command:

```
create_field -name AbsSumElectricField -dataset 2D
   -function "abs(<ElectricField-X>+<ElectricField-Y>)" -show
```

NOTE New fields also can be created on the **More** tab of the Selection panel by clicking the **Add Field** button.

Available Functions

Table 28 on page 318 lists the available functions. The function arguments are:

- *Double*: Numeric values, scalar field names.
- *Vector*: Vector field names.
- *Curve*: 1D curve names.

For example:

```
-function "sin(<ElectricField-X>+<ElectricField-Y>)" (Double or Scalar)

-function "sin(<ElectricField-V>)" (Vector)

-function "sin(<Curve_1>)" (Curve)
```

Table 28 Available functions

Function	Arguments	Returns	Description
abs(x)	Double Vector Curve	Double Vector Curve	Absolute value.
acos(x)	Double Vector Curve	Double Vector Curve	ArcCosine.
acosh(x)	Double Vector Curve	Double Vector Curve	Hyperbolic ArcCosine.
asin(x)	Double Vector Curve	Double Vector Curve	ArcSine.
asinh(x)	Double Vector Curve	Double Vector Curve	Hyperbolic ArcSine.
atan(x)	Double Vector Curve	Double Vector Curve	ArcTangent.
atanh(x)	Double Vector Curve	Double Vector Curve	Hyperbolic ArcTangent.
bessel_j0(x)	Double Vector Curve	Double Vector Curve	Bessel function of first kind, order zero.
bessel_j1(x)	Double Vector Curve	Double Vector Curve	Bessel function of first kind, first order.
bessel_y0(x)	Double Vector Curve	Double Vector Curve	Bessel function of second kind, order zero.
bessel_y1(x)	Double Vector Curve	Double Vector Curve	Bessel function of second kind, first order.
cbrt(x)	Double Vector Curve	Double Vector Curve	Cube root of x.
ceil(x)	Double Vector Curve	Double Vector Curve	Approximates to the next integer.

Table 28 Available functions

Function	Arguments	Returns	Description
cfftim(x,y)	x: Vector, Curve y: Vector, Curve	Vector Curve	Fast Fourier transform, imaginary value.
cfftre(x,y)	x: Vector, Curve y: Vector, Curve	Vector Curve	Fast Fourier transform, real value.
cifftim(x,y)	x: Vector, Curve y: Vector, Curve	Vector Curve	Inverse fast Fourier transform, imaginary value.
cifftre(x,y)	x: Vector, Curve y: Vector, Curve	Vector Curve	Inverse Fourier transform, real value.
cos(x)	Double Vector Curve	Double Vector Curve	Cosine.
cosh(x)	Double Vector Curve	Double Vector Curve	Hyperbolic cosine.
crop(x, y, min, max) crop(c, min, max)	x: Vector y: Vector min: Double max: Double c: Curve	Vector Curve	Crops the values depending of the minimum and maximum range of x.
diff(c) diff(y,x)	x: Vector y: Vector c: Curve	Vector Curve	First-order derivative.
erf(x)	Double Vector Curve	Double Vector Curve	Error function.
erfc(x)	Double Vector Curve	Double Vector Curve	Complementary error function.
exp(x)	Double Vector Curve	Double Vector Curve	Evaluates $e^{(x)}$.
fftabs(y,x)	x: Vector, Curve y: Vector, Curve	Vector Curve	Fast Fourier transform, absolute value.
fftim(x)	Vector Curve	Vector Curve	Fast Fourier transform, imaginary value.
fftre(x)	Vector Curve	Vector Curve	Fast Fourier transform, real value.

Table 28 Available functions

Function	Arguments	Returns	Description
floor(x)	Double Vector Curve	Double Vector Curve	Approximates to the previous integer.
gamma(x)	Double Vector Curve	Double Vector Curve	Gamma function.
ifftim(x)	Vector Curve	Vector Curve	Inverse Fourier transform, imaginary value.
ifftre(x)	Vector Curve	Vector Curve	Inverse Fourier transform, real value.
integr(y,x) integr(c)	y: Vector x: Vector c: Curve	Vector Curve	Integrates the vector y over the range specified by x, or integrates the curve c.
inverse(x)	Double Vector Curve	Double Vector Curve	Inverse value.
lgamma(x)	Double Vector Curve	Double Vector Curve	Logarithmic gamma function.
log(x)	Double Vector Curve	Double Vector Curve	Natural logarithm.
log10(x)	Double Vector Curve	Double Vector Curve	Common logarithm.
pow(x,y)	x: Double x: Vector x: Curve y: Double	Double Vector Curve	Evaluates xy, where x is a double value, a vector of values, or a curve.
rms(x,y)	x: Vector x: Curve y: Vector y: Curve	Double	Root mean square value.
sign(x)	Double Vector Curve	Double Vector Curve	Sign.
sin(x)	Double Vector Curve	Double Vector Curve	Sine.

Table 28 Available functions

Function	Arguments	Returns	Description
sinh(x)	Double Vector Curve	Double Vector Curve	Hyperbolic sine.
sqrt(x)	Double Vector Curve	Double Vector Curve	Square root.
tan(x)	Double Vector Curve	Double Vector Curve	Tangent.
tangent(c,v) tangent(x,y,v)	x: Vector y: Vector c: Curve v: Double	Vector Curve	Creates a tangent line in the point v on the curve c, or the curve defined by the vectors x and y.
tanh(x)	Double Vector Curve	Double Vector Curve	Hyperbolic tangent.
vecmax(x) vecmax(c)	x: Vector c: Curve	Double	Returns the maximum y-value of a curve, or the maximum value of a vector.
vecmin(x) vecmin(c)	x: Vector c: Curve	Double	Returns the minimum y-value of a curve, or the minimum value of a vector.
vecvalx(x,y,v) vecvalx(c,v)	x: Vector y: Vector c: Curve v: Double	Double	Returns the x-value when y=v of a curve.
vecvaly(x,y,v) vecvaly(c,v)	x: Vector y: Vector c: Curve v: Double	Double	Returns the y-value when x=v of a curve.
veczero(x,y) veczero(c)	x: Vector y: Vector c: Curve	Double	Returns the x-value when y=0 of a curve.

NOTE For functions that only return a Double value and are used as the outer function in formulas, the result will be displayed only in a dialog box and cannot be used in Tcl scripts.

APPENDIX D Inspect Support in Sentaurus Visual

This appendix provides information about the level of support for running Inspect scripts in Sentaurus Visual.

The support of Inspect commands is available only if the commands are in a script file and it is loaded using one of the following options:

- From the command line, for example, pass an Inspect script file as an argument with the corresponding option:

  ```
  svisual -inspect test_ins.cmd
  ```

- From the user interface, choose **File > Run Tcl Script**. In the Open Script File dialog box, select **Inspect Command File (*.cmd)** in the **Files of type** field.
- Load a Tcl script using the `load_script_file` Tcl command, for example:

  ```
  load_script_file test_ins.cmd -inspect
  ```

Fully Supported Commands

The following Inspect commands are fully supported in this version of Sentaurus Visual:

- `load_library`
- `ft_scalar`
- `proj_getDataSet`
- `proj_getList`
- `proj_getNodeList`
- `proj_load`
- `proj_unload`
- `cv_create`
- `cv_createDS`
- `cv_createFromScript`
- `cv_createWithFormula`
- `cv_delete`
- `cv_display`

- `cv_logScale`
- `cv_log10Scale`
- `cv_split`
- `cv_split_disc`
- `cv_lineColor`
- `cv_lineStyle`
- `gr_createLabel`
- `gr_mappedAxis`
- `gr_setGridAttr`
- `cv_getVals`
- `cv_getValsX`
- `cv_getValsY`
- `cv_getXaxis`
- `cv_getYaxis`
- `cv_printVals`
- `cv_abs`
- `f_Gamma`
- `f_gm`
- `f_IDSS`
- `f_KP`
- `f_Ron`
- `f_Rout`
- `f_TetaG`
- `f_VT`
- `cv_compute`
- `cv_getZero`
- `macro_define`
- `script_exit`
- `script_sleep`
- `gr_formatAxis`
- `gr_precision`
- `gr_setLegend`
- `gr_setLegendPos`

Partially Supported Commands

Sentaurus Visual only partially supports the Inspect commands listed in Table 29.

Table 29 Partially supported Inspect commands

Command	Limitations
`cv_renameCurve`	Works only if the curve is not displayed.
`cv_set_interpol`	Works only if the curve is displayed.
`cv_setCurveAttr`	Cannot set color and width of the marker outline. Cannot set the fill color of the marker. Triangle marker is not available.
`gr_setAxisAttr`	Cannot set color and width of the axis line. Cannot set number of secondary ticks and angle at which the tick labels are drawn.
`gr_setGeneralAttr`	Only background color can be set.
`gr_setLegendAttr`	Cannot set frame color, width position, and anchor.
`gr_setTitleAttr`	Cannot set title justification.
`script_break`	Suspends the script, displaying a message.

Not Supported Commands

The following Inspect commands are not supported in this version of Sentaurus Visual:

- `cv_write`
- `fi_writeBitmap`
- `fi_writeEps`
- `fi_writePs`
- `graph_load`
- `graph_write`
- `param_load`
- `param_write`
- `proj_write`
- `gb_setpreferences`
- `gr_deleteLabel`
- `cv_delPts`

- `cv_inv`
- `cv_reset`
- `f_hideInternalCurves`
- `f_showInternalCurves`
- `f_VT1`
- `f_VT2`

Script Library Support

This section explains the support Sentaurus Visual provides for different Inspect script libraries.

Extraction Library

All the commands from this library are fully supported only if they are calculated over displayed curves:

- `ExtractEarlyV`
- `ExtractGm`
- `ExtractGmb`
- `ExtractIoff`
- `ExtractMax`
- `ExtractRon`
- `ExtractSS`
- `ExtractValue`
- `ExtractVtgm`
- `ExtractVtgmb`
- `ExtractVti`

If the curve is created but not displayed, the result will be the same for all commands except `ExtractIoff` because the interpolation will be linear not logarithmic.

Curve Comparison Library

Both commands generate a new curve with specific visual properties of the marker, which are not necessarily the same as in Inspect, that is, there is a visual difference:

- `cvcmp_CompareTwoCurves`
- `cvcmp_DeltaTwoCurves`

The extend Library

Partially Supported Commands

Sentaurus Visual only partially supports the commands listed in Table 30.

Table 30 Partially supported commands of extend library

Command	Limitations
`cv_autoIncrStyle` `cv_disp`	Depends on `cv_setCurveAttr`, which is not fully supported. For more details on the limitations of `cv_setCurveAttr`, see Table 29 on page 325.
`cv_nextSymbol` `cv_setSymbol`	Triangle marker is not available.

Not Supported Commands

The following commands are not supported in this version of Sentaurus Visual:

- `cv_exists`
- `cv_resetFillColor`
- `cv_setFillColor`
- `ds_getValue`
- `proj_check`
- `proj_datasetExists`
- `proj_getGroups`
- `proj_groupExists`

APPENDIX E Extraction Library

This appendix provides information about the procedures of the extraction library.

The procedures of the extraction library are used to extract various parameters from the I–V characteristics of various device types. The extraction library takes I–V data in the form of two Tcl lists: one list contains the voltages points and the other list contains the corresponding current values.

The extraction library is loaded automatically when Sentaurus Visual starts. However, if you have disabled the automatic loading of extension libraries, you can load the extraction library explicitly with the command:

```
load_library extract
```

Syntax Conventions

The extraction library uses a unique namespace identifier (`ext::`) for its procedures. All procedures and variables associated with this library are called with the namespace identifier prepended, for example:

```
ext::<proc_name>
```

Each procedure has several arguments. The extraction library uses an input parser that accepts arguments of the form:

```
-keyword <value>
```

NOTE All Sentaurus Visual libraries support the standard Sentaurus Visual syntax in which keywords are preceded by a dash. For backward compatibility, all Sentaurus Visual libraries continue to support the `keyword= <value>` syntax as well. For each procedure call, you can use either the `-keyword <value>` syntax or the `keyword= <value>` syntax. However, within any one procedure call, only one type of syntax can be used. Only the new syntax is documented. If you want to continue using the `keyword= <value>` syntax, you also can insert whitespace between the keyword and the equal sign, for example, `keyword = <value>`. Omitting the whitespace between the equal sign and the value field will result in a failure if the value is a de-referenced Tcl variable. Use `keyword= $val` (*not* `keyword=$val`).

The parser accepts arguments in any order. For some arguments, default values are predefined. Such arguments may be omitted. If arguments for which no defaults are predefined are omitted, the procedure will exit with an error message. In addition, unrecognized arguments result in an error message.

Instead of using the standard Tcl method of using the return value of the procedure to pass results back to the calling program, the extraction library uses a *passing-by-reference* method to return the results to the calling program. The procedure keyword `-out` is used to pass the results back to the calling program:

```
-out <var_name>, <list_name>, or <array_name>
```

The following conventions are used for the syntax of Tcl commands:

- Angle brackets – <> – indicate text that must be replaced, but they are *not* part of the syntax. In particular, the following type identifiers are used:
  - `<r>`: Replace with a real number, or a de-referenced Tcl variable that evaluates to a real number. For example: `$val`.
  - `<i>`: Replace with an integer, or a de-referenced Tcl variable that evaluates to an integer. For example: `$i`.
  - `<string>`: Replace with a string, or a de-referenced Tcl variable that evaluates to a string. For example: `$file`.
  - `<list_of_r>`: Replace with a list of real numbers, or a de-referenced Tcl variable that evaluates to a list of real numbers. For example: `$values`.
  - `<list_of_strings>`: Replace with a list of strings, or a de-referenced Tcl variable that evaluates to a list of strings. For example: `$files`.
  - `<var_name>`: Replace with the *name* of a local Tcl variable.
    For example: `val` (*not* `$val`).
  - `<list_name>`: Replace with the *name* of a local Tcl list.
    For example: `values` (*not* `$values`).
  - `<array_name>`: Replace with the *name* of a local Tcl array.
    For example: `myarray` (*not* `$myarray`).
- Brackets – [] – indicate that the argument is optional, but they are *not* part of the syntax.
- A vertical bar – | – indicates options, only one of which can be specified.

Help for Procedures

To request help on a specific procedure, set the `-help` keyword to `1`:

```
ext::<proc_name> -help 1
```

If this command is included in a Sentaurus Visual file, when Sentaurus Visual is executed in:

- Batch mode in Sentaurus Workbench, the help information is printed to the runtime output file (with the extension `.out`) of the corresponding Sentaurus Visual node.
- Interactive mode in Sentaurus Workbench, the help information is displayed in the Tcl Command panel as well as printed in the Sentaurus Visual output file.

You also can type this command in the Tcl Command panel of the graphical user interface, in which case, the help information is displayed in the same panel.

Output of Procedures

As discussed in Syntax Conventions on page 329, all procedures of the extraction library pass the results back to the calling program by storing the results in a Tcl variable. The name of this Tcl variable is specified as the value of the `-out` keyword. All procedures beginning with `ext::Extract` extract a device parameter. For example, the procedure `ext::ExtractVtgm` extracts the threshold voltage:

```
ext::ExtractVtgm -out Vt -name Vtgm -v $Vgs -i $absIds
```

Here, since `-out Vt` is used, the extracted threshold voltage is stored in the Tcl variable `Vt`.

All procedures of the extraction library beginning with `ext::Extract` pass the extracted value to the Sentaurus Workbench Family Tree (if the `-name` keyword differs from `"noprint"`). The extracted quantity is displayed as a Sentaurus Workbench variable.

If `-name "noprint"` is used, the extracted variable is not passed to the Sentaurus Workbench Family Tree. If `-name out` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree.

Here, since `-name Vtgm` is used, the extracted threshold voltage value is displayed as the Sentaurus Workbench variable `Vtgm`.

If there are errors in the procedures, the behavior of Sentaurus Visual depends on whether it is executed in batch mode or interactive mode in Sentaurus Workbench. In batch mode, Sentaurus Visual exits and an error message is printed only in the Sentaurus Visual error file (with the

extension `.err`). In interactive mode, the error message is displayed in the Tcl Command panel as well as printed in the Sentaurus Visual error file.

All procedures also print several messages (including warning messages). If Sentaurus Visual is executed in batch mode, the messages are printed only in the Sentaurus Visual output file; whereas, in interactive mode, the messages are displayed in the Tcl Command panel as well as printed in the Sentaurus Visual output file.

The amount of information printed depends on the information level specified by the procedure `lib::SetInfoDef`. Irrespective of the specified information level, the extracted value is printed in the output file by the procedures beginning with `ext::Extract`.

For example, if the information level is set to 0 for all procedures using the `lib::SetInfoDef` procedure:

```
lib::SetInfoDef 0
ext::ExtractVtgm -out Vt -name Vtgm -v $Vgs -i $absIds
```

the following message is printed:

```
DOE: Vtgm 0.316
```

If the information level for the procedure `ext::ExtractVtgm` is set to 1 using the `lib::SetInfoDef` procedure:

```
lib::SetInfoDef 1
ext::ExtractVtgm -out Vt -name Vtgm -v $Vgs -i $Ids -vo 1e-4
```

or by using the `-info` keyword:

```
lib::SetInfoDef 0
ext::ExtractVtgm -out Vt -name Vtgm -v $Vgs -i $Ids -vo 1e-4 -info 1
```

the following message is printed:

```
DOE: Vtgm 0.316
Vtgm (Max gm method): 0.316
```

If the extraction library procedure cannot extract the parameter, the parameter is set to the character ‘`x`’ and a message is printed. In the case of `ext::ExtractVtgm`, the following message is printed:

```
DOE: Vtgm x
ext::ExtractVtgm: Vtgm not found!
```

ext::AbsList

Computes the absolute value of all elements of a list.

Syntax

```
ext::AbsList -out <list_name> -x <list_of_r>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <list_name>`	Name of a list to store the list of absolute values. (List name, no default)
`-x <list_of_r>`	Input list. (List of real numbers, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVg_des.plt -name DC
```

```
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]
puts "Ids= $Ids"
ext::AbsList -out Idabs -x $Ids
puts "Idabs= $Idabs"
#-> Ids= -1.42055e-08 -3.64403e-08 -9.11723e-08 ... -3.6233e-5
#-> Idabs= 1.42055e-08 3.64403e-08 9.11723e-08 ... 3.6233e-5
```

ext::DiffForwardList

Computes the first-order derivative of a curve using the forward finite difference method. The curve is represented by two Tcl lists: one contains the x-values (independent variable) and one contains the corresponding y-values (dependent variable).

Syntax

```
ext::DiffForwardList -out <array_name> -x <list_of_r> -y <list_of_r>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements `x` and `dy`. The values of the `x` element and the `dy` element are lists of x-values and first-order derivatives, respectively. (Array name, no default)
`-x <list_of_r>`	List containing the x-values (independent variable). (List of real numbers, no default)
`-y <list_of_r>`	List containing the y-values (dependent variable). (List of real numbers, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Xs [list 1.0 2.0 3.0 4.0]
# Generate Ys using y=2*x
set Ys [list]
foreach x $Xs {
   lappend Ys [expr 2*$x]
}
ext::DiffForwardList -out DyDx -x $Xs -y $Ys
puts "x-values= $DyDx(x)"
puts "y-values= $Ys"
puts "derivative= $DyDx(dy)"
#-> x-values  = 1.5 2.5 3.5
#-> y-values  = 2.0 4.0 6.0 8.0
#-> derivative= 2.0 2.0 2.0
```

ext::DiffList

Computes the first-order derivative of a curve. The curve is represented by two Tcl lists: one contains the x-values (independent variable) and one contains the corresponding y-values (dependent variable).

NOTE The procedure `ext::DiffList` uses the central finite difference method to compute the derivative at a data point. This method uses the x- and y-values of two adjacent points, which are computed internally by the procedure using either linear or logarithmic interpolation.

Syntax

```
ext::DiffList -out <list_name> -x <list_of_r> -y <list_of_r>
   [-yLog 0 | 1] [-xLog 0 | 1] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <list_name>`	Name of a list to store the first-order derivative. (List name, no default)
`-x <list_of_r>`	List containing the x-values (independent variable). (List of real numbers, no default)
`-y <list_of_r>`	List containing the y-values (dependent variable). (List of real numbers, no default)
`-yLog 0 | 1`	Selects linear (`0`) or logarithmic (`1`) interpolation for y-axis values for computing the derivative. Default: `0`
`-xLog 0 | 1`	Selects linear (`0`) or logarithmic (`1`) interpolation for x-axis values for computing the derivative. Default: `0`
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Xs [list 1 1.5 2.5 6 7 7.5 8.5 8.7 8.8 10]
# Generate Ys using y=exp(x)+1
set Ys [list]
foreach x $Xs {
   lappend Ys [expr exp($x) + 1]
}
puts "y= $Ys"
# For exponential function, use logarithmic interpolation for y-axis values
set yLog 1
ext::DiffList -out dydx -x $Xs -y $Ys -yLog $yLog
puts "dydx= $dydx"
#-> y= 3.718 5.481 13.182 ... 22027.465
#-> dydx= 2.887 4.532 12.231 ... 22012.323
```

ext::ExtractBVi

Extracts the breakdown voltage from an I–V curve. The breakdown voltage is defined as the bias voltage at which the current reaches a certain level. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractBVi -out <var_name> -v <list_of_r> -i <list_of_r> -io <r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the breakdown voltage. (Real number, no default)
`-v <list_of_r>`	List containing the voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the current values. (List of real numbers, no default)
`-io <r>`	Current level. (Real number, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"BVi"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3e"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IcVc_des.plt -name BV

set Vcs [get_variable_data "collector InnerVoltage" -dataset BV]
set Ics [get_variable_data "collector TotalCurrent" -dataset BV]

ext::ExtractBVi -out BVcboi -name "out" -v $Vcs -i $Ics -io 1e-12
puts "BVi is [format %.3e $BVcboi] V"
#-> BVi is 9.199e+00 V
```

ext::ExtractBVv

Extracts the breakdown voltage from an I–V curve. The breakdown voltage is defined as the maximum voltage that can be applied to a contact. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractBVv -out <var_name> -v <list_of_r> -i <list_of_r> -sign <+1 | -1>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the breakdown voltage. (Real number, no default)
`-v <list_of_r>`	List containing the voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the current values. (List of real numbers, no default)
`-sign <+1 | -1>`	Distinguishes different types of bipolar transistor: `+1`: n-p-n transistor `-1`: p-n-p transistor In general, set `-sign` to `-1` if the breakdown occurs at a negative bias. (No default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"BVv"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.2e"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IcVc_des.plt -name BV

set Vcs [get_variable_data "collector InnerVoltage" -dataset BV]
set Ics [get_variable_data "collector TotalCurrent" -dataset BV]

ext::ExtractBVv -out BVcbov -name "out" -v $Vcs -i $Ics -sign 1
puts "BVv is [format %.2e $BVcbov] V"
#-> BVv is 9.20e+00 V
```

ext::ExtractEarlyV

Extracts the Early voltage from an I_c–V_{ce} curve. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractEarlyV -out <var_name> -v <list_of_r> -i <list_of_r> -vo <r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the Early voltage. (Real number, no default)
`-v <list_of_r>`	List containing the voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the collector current values. (List of real numbers, no default)
`-vo <r>`	Bias point at which the slope of the I_c–V_{ce} curve is determined for the computation of the Early voltage. (Real number, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Va"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3e"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Extract Early voltage for a p-n-p bipolar transistor
load_file IcVc_des.plt -name IcVce

set Vcs [get_variable_data "collector OuterVoltage" -dataset IcVce]
set Ics [get_variable_data "collector TotalCurrent" -dataset IcVce]
ext::AbsList -out absIcs -x $Ics ;# Compute absolute value of collector current

ext::ExtractEarlyV -out Va -name "out" -v $Vcs -i $absIcs -vo -1.25
puts "Early Voltage is [format %.3e $Va] V"
#-> Early Voltage is 1.897e+01 V
```

ext::ExtractExtremum

Extracts the maximum or minimum of a curve. The curve is represented by two Tcl lists: one contains the x-values and one contains the corresponding y-values.

Syntax

```
ext::ExtractExtremum -out <var_name> -x <list_of_r> -y <list_of_r>
   [-type "max" | "min"] [-name <string>] [-f <string>]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the maximum or minimum of the curve. (Real number, no default)
`-x <list_of_r>`	List containing the x-values. (List of real numbers, no default)
`-y <list_of_r>`	List containing the y-values. (List of real numbers, no default)
`-type "max" | "min"`	Selects whether to extract the minimum (`"min"`) or maximum (`"max"`) of a curve. Default: `"max"`
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"out"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3e"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVg_des.plt -name DC
```

```
set Vgs [get_variable_data "gate OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]
```

```
ext::ExtractExtremum -out IdSat -name "out" -x $Vgs -y $Ids -type "max"
puts "IdSat is [format %.3e $IdSat] A/um"
#-> IdSat is 4.028e-4 A/um
```

ext::ExtractGm

Extracts the maximum transconductance from an I_d–V_{gs} curve. The transconductance g_m is defined as:

$$g_m = \frac{dI_d}{dV_g} \tag{1}$$

The gate bias at which the maximum transconductance occurs is computed using parabolic interpolation. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractGm -out <var_name> -v <list_of_r> -i <list_of_r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the maximum transconductance. (Real number, no default)
`-v <list_of_r>`	List containing the gate voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the drain current values. (List of real numbers, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"gm"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3e"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Extract gm for a PMOSFET
load_file IdVg_des.plt -name DC

set Vgs [get_variable_data "gate OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]
ext::AbsList -out absIds -x $Ids

ext::ExtractGm -out gm -name "out" -v $Vgs -i $absIds
puts "gm is [format %.3e $gm] S/um"
#-> gm is 6.780e-05 S/um
```

ext::ExtractIoff

Extracts the drain leakage current at the specified gate voltage from an I_d–V_{gs} curve (computed for a high drain bias). The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values. The drain leakage current is extracted at a small nonzero gate voltage value to avoid noise.

Syntax

```
ext::ExtractIoff -out <var_name> -v <list_of_r> -i <list_of_r> -vo <r>
   [-log10 0 | 1] [-name <string>] [-f <string>]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the drain leakage current. (Real number, no default)
`-v <list_of_r>`	List containing the gate voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the drain current values. (List of real numbers, no default)
`-vo <r>`	Gate voltage at which the drain leakage current is extracted. It is recommended to use a small but nonzero value, such as 0.1 mV for an NMOS device and –0.1 mV for a PMOS device. (Real number, no default)
`-log10 0 | 1`	Procedure returns log10(I_{off}) if set to `1`. Otherwise, the procedure returns I_{off}. Default: `0`
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Ioff"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3e"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVg_des.plt -name DC
set Vgs [get_variable_data "gate OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]

ext::ExtractIoff -out Ioff -name "out" -v $Vgs -i $Ids -vo 1e-4 -log10 0
puts "Ioff is [format %.3e $Ioff] A/um"

ext::ExtractIoff -out log10Ioff -name "out" -v $Vgs -i $Ids -vo 1e-4 -log10 1
puts "Log10Ioff is [format %.3e $log10Ioff]"
#-> Ioff is 1.151e-7 A/um
#-> Log10Ioff is -6.939e+00
```

ext::ExtractRdiff

Extracts the differential resistance R_{diff} from an I–V curve at a specified voltage. R_{diff} is defined as:

$$R_{\text{diff}} = \frac{dV}{dI} \tag{2}$$

The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractRdiff -out <var_name> -v <list_of_r> -i <list_of_r> -vo <r>
   [-yLog 0 | 1] [-xLog 0 | 1] [-name <string>] [-f <string>]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the differential resistance. (Real number, no default)
`-v <list_of_r>`	List containing the voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the current values. (List of real numbers, no default)
`-vo <r>`	Voltage at which the differential resistance is extracted. (Real number, no default)
`-yLog 0 | 1`	Selects linear (`0`) or logarithmic (`1`) interpolation for y-axis values for computing the derivative. See note in ext::DiffList on page 335. Default: `0`
`-xLog 0 | 1`	Selects linear (`0`) or logarithmic (`1`) interpolation for x-axis values for computing the derivative. See note in ext::DiffList on page 335. Default: `0`
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Rdiff"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3f"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Extract on-state output resistance of a p-n-p bipolar transistor
load_file IcVc_des.plt -name IcVce

set Vcs [get_variable_data "collector OuterVoltage" -dataset IcVce]
set Ics [get_variable_data "collector TotalCurrent" -dataset IcVce]

ext::AbsList -out absIcs -x $Ics
ext::ExtractRdiff -out Ron -name "out" -v $Vcs -i $absIcs -vo -1.25
puts "Ron is [format %.3e $Ron] Ohm-um"
#-> Ron is 38077.106 Ohm um
```

ext::ExtractRsh

Calculates the sheet resistance $R_{sh}[\Omega/\text{sq}]$ and the p-n junction depth of semiconductor layers in the vertical direction in a 2D structure by creating an axis-aligned cutline. It also calculates the total sheet resistance (the sum of the sheet resistance of each layer).

NOTE This procedure applies only to 2D structures.

The sheet resistance of each semiconductor layer is computed using:

$$R_{sh} = \frac{1}{\int_0^d \sigma(x)dx} \tag{3}$$

where d is the thickness of the semiconductor layer, and σ is its conductivity given by:

$$\sigma(x) = q[n(x)\mu_n(x) + p(x)\mu_p(x)] \tag{4}$$

where:

- q is the elementary charge.
- n and p are the electron and hole density, respectively.
- μ_n and μ_p are the electron and hole mobility, respectively.

The resistivity ρ is given by:

$$\rho(x) = \frac{1}{\sigma(x)} \tag{5}$$

NOTE The axis-aligned cutline is created using the `create_cutline` command (see create_cutline on page 155).

Syntax

```
ext::ExtractRsh -out <array_name> -dataset <dataName> -semdataset <dataName>
   -type <x | y> -at <r>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements `d`, `Rsh`, `RshTotal`, `RshTop`, `xjTop`, and other elements. The values of the elements `d` and `Rsh` are the thickness and sheet resistance of each semiconductor layer. The values of the elements `RshTotal`, `RshTop`, and `xjTop` are the total sheet resistance, the sheet resistance of the top semiconductor layer, and the junction depth, respectively. The index also contains the elements `X` (for `-type y`) or `Y` (for `-type x`), `DopingConcentration`, `eMobility`, `hMobility`, `eDensity`, `hDensity`, `Conductivity`, and `Resistivity`. The values of the elements `X` and `Y` are lists of x-axis values and y-axis values, respectively. The values of the elements `DopingConcentration`, `eMobility`, `hMobility`, `eDensity`, `hDensity`, `Conductivity`, and `Resistivity` are lists of the doping concentration, electron mobility, hole mobility, electron density, hole density, conductivity, and resistivity, respectively. (Array name, no default)
`-dataset <dataname>`	Name of the dataset from where the cutline will be generated. (String, no default)
`-semdataset <dataname>`	Name of the dataset containing the variables `X` (for `-type y`) or `Y` (for `-type x`), `DopingConcentration`, `eMobility`, `hMobility`, `eDensity`, `hDensity`, `Conductivity`, and `Resistivity`. These variables contain a list of x-axis values (for `-type y`), or y-axis values (for `-type x`), doping concentration, electron mobility, hole mobility, electron density, hole density, conductivity, and resistivity, respectively. (String, no default)
`-type x | y`	Links the cutline to the specified axis. If `-type x` (`-type y`) is specified, the cutline is linked to the x-axis (y-axis), and the cutline is created axis-aligned to the y-axis (x-axis). Specify `-type` so that the cutline is created in the vertical direction. (String, no default)
`-at <r>`	Value where the axis-aligned cutline cuts the axis to which it is linked. (Real number, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file LDMOS_des.tdr -name Structure2d
create_plot -name Plot_Structure2d -dataset Structure2d
# Create cutline y=5.0
ext::ExtractRsh -out Rsh -dataset Structure2d -semdataset semdata \
   -type y -at 5.0
# Plot conductivity profile
create_plot -1d -name Plot_Profile
create_curve -dataset semdata -axisX X -axisY Conductivity
# Extract sheet resistance of top layer
puts "DOE: Rshtop [format %.2f $Rsh(RshTop)]"
# Extract p-n junction depth
puts "DOE: xj [format %.3f $Rsh(xjTop)]"
```

ext::ExtractSS

Extracts the subthreshold voltage swing, for a given gate voltage V_{go}, from an I_d–V_{gs} curve. The subthreshold voltage swing (SS) is defined as:

$$SS = \frac{1000}{\frac{d}{dV_g}\log_{10} I_d} \tag{6}$$

where V_g is given in V, I_d is given in A/µm or A, and SS is given in mV/decade. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

NOTE The slope may be *noisy* at the beginning of the curve or at very low current levels. Better results are often obtained when setting V_{go} to a small but nonzero value.

Syntax

```
ext::ExtractSS -out <var_name> -v <list_of_r> -i <list_of_r> -vo <r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the subthreshold voltage swing. (Real number, no default)
`-v <list_of_r>`	List containing the gate voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the absolute value of the drain currents. (List of real numbers, no default)
`-vo <r>`	Gate voltage at which the slope is extracted. It should be a value well below the threshold voltage. (Real number, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"SS"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3f"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Vgo 1e-2
load_file IdVg_des.plt -name DC

set Vgs [get_variable_data "gate OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]

ext::ExtractSS -out SS -name "out" -v $Vgs -i $Ids -vo $Vgo
puts "SS (subthreshold voltage swing) is [format %.3f $SS] mV/dec"
#-> SS (subthreshold voltage swing) is 89.555 mV/dec
```

ext::ExtractSsub

Extracts the subthreshold voltage swing from an I_d–V_{gs} curve. The subthreshold voltage swing (SS) is defined as:

$$SS = \frac{1000}{Max\left(\frac{d}{dV_g}\log_{10} I_d\right)} \tag{7}$$

where V_g is given in V, I_d is given in A/µm or A, SS is given in mV/decade, and $Max\left(\frac{d}{dV_g}\log_{10} I_d\right)$ is the maxima of $\frac{d}{dV_g}\log_{10} I_d$.

The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractSsub -out <var_name> -v <list_of_r> -i <list_of_r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the subthreshold voltage swing. (Real number, no default)
`-v <list_of_r>`	List containing the gate voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the absolute value of the drain currents. (List of real numbers, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Ssub"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3f"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVg_des.plt -name DC

set Vgs [get_variable_data "gate OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]

ext::ExtractSsub -out Ssub -name "out" -v $Vgs -i $Ids
puts "Ssub (subthreshold voltage swing) is [format %.3f $Ssub] mV/dec"
#-> Ssub (subthreshold voltage swing) is 82.786 mV/dec
```

ext::ExtractValue

For a given target x-value, the procedure extracts the *n* -th interpolated y-value in a curve. The curve is represented by two Tcl lists: one contains the x-values and one contains the corresponding y-values.

NOTE To find the interpolated x-value for a given y-value, swap the arguments `x` and `y`.

Syntax

```
ext::ExtractValue -out <var_name> -x <list_of_r> -y <list_of_r> -xo <r>
   [-occurrence <i>] [-yLog 0 | 1] [-xLog 0 | 1] [-name <string>]
   [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the first found y-value. (Real number, no default)
`-x <list_of_r>`	List containing the x-values. (List of real numbers, no default)
`-y <list_of_r>`	List containing the y-values. (List of real numbers, no default)
`-xo <r>`	Target x-value. (Real number, no default)
`-occurrence <i>`	Specifies the *n* -th interpolated y-value to be extracted. (Integer, default: 1)
`-yLog 0 | 1`	Selects linear (`0`) or logarithmic (`1`) interpolation for y-axis values. Default: `0`
`-xLog 0 | 1`	Selects linear (`0`) or logarithmic (`1`) interpolation for x-axis values. Default: `0`
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"out"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3e"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Xs [list 1e2 1e3 1e4 1e5 1e6 1e7 1e8 1e9 1e10 1e11 1e12]
set Ys [list -100 -25 25 50 100 100 75 50 25 -25 -100]

ext::ExtractValue -out Flog -x $Ys -y $Xs -yLog 1 -xo 0
puts "Flog= [format %.3e $Flog]"
ext::ExtractValue -out Flin -x $Ys -y $Xs -yLog 0 -xo 0
puts "Flin= [format %.3e $Flin]"
ext::ExtractValue -out F2nd -x $Ys -y $Xs -yLog 1 -xo 0 -occurrence 2
puts "F2nd= [format %.3e $F2nd]"

#-> Flog= 3.162e+03
#-> Flin= 5.500e+03
#-> F2nd= 3.162e+10
```

ext::ExtractVdlin

Extracts the drain voltage for a given drain current level from an I_d–V_{ds} curve using linear interpolation. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractVdlin -out <var_name> -v <list_of_r> -i <list_of_r> -io <r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the drain voltage corresponding to the specified drain current level. (Real number, no default)
`-v <list_of_r>`	List containing the drain voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the drain current values. (List of real numbers, no default)
`-io <r>`	Drain current level. (Real number, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Vdlin"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3f"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVd_des.plt -name DC

set Vds [get_variable_data "drain OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]

ExtractVdlin -out Vdlin -name "out" -v $Vds -i $Ids -io 1e-4
puts "Vdlin (Vd at Io= 1e-4 A/um) is [format %.3f $Vdlin] V"
#-> Vdlin (Vd at Io= 1e-4 A/um) is 0.046 V
```

ext::ExtractVdlog

Extracts the drain voltage for a given drain current level from an I_d–V_{ds} curve using logarithmic interpolation. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractVdlog -out <var_name> -v <list_of_r> -i <list_of_r> -io <r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the drain voltage corresponding to the specified drain current level. (Real number, no default)
`-v <list_of_r>`	List containing the drain voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the drain current values. (List of real numbers, no default)
`-io <r>`	Drain current level. (Real number, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Vdlog"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3f"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVd_des.plt -name DC

set Vds [get_variable_data "drain OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]

ExtractVdlog -out Vdlog -name "out" -v $Vds -i $Ids -io 1e-4
puts "Vdlog (Vd at Io= 1e-4 A/um) is [format %.3f $Vdlog] V"
#-> Vdlog (Vd at Io= 1e-4 A/um) is 0.050 V
```

ext::ExtractVglin

Extracts the gate voltage for a given drain current level from an I_d–V_{gs} curve using linear interpolation. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractVglin -out <var_name> -v <list_of_r> -i <list_of_r> -io <r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the gate voltage corresponding to the specified drain current level. (Real number, no default)
`-v <list_of_r>`	List containing the gate voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the drain current values. (List of real numbers, no default)
`-io <r>`	Drain current level. (Real number, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Vglin"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3f"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVg_des.plt -name DC

set Vgs [get_variable_data "gate OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]

ExtractVglin -out Vglin -name "out" -v $Vgs -i $Ids -io 1e-7
puts "Vglin (Vg at Io= 1e-7 A/um) is [format %.3f $Vglin] V"
#-> Vglin (Vg at Io= 1e-7 A/um) is 0.137 V
```

ext::ExtractVglog

Extracts the gate voltage for a given drain current level from an I_d–V_{gs} curve using logarithmic interpolation. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractVglog -out <var_name> -v <list_of_r> -i <list_of_r> -io <r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the gate voltage corresponding to the specified drain current level. (Real number, no default)
`-v <list_of_r>`	List containing the gate voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the drain current values. (List of real numbers, no default)
`-io <r>`	Drain current level. (Real number, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Vglog"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3f"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVg_des.plt -name DC

set Vgs [get_variable_data "gate OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]

ExtractVglog -out Vglog -name "out" -v $Vgs -i $Ids -io 1e-7
puts "Vglog (Vg at Io= 1e-7 A/um) is [format %.3f $Vglog] V"
#-> Vglog (Vg at Io= 1e-7 A/um) is 0.142 V
```

ext::ExtractVtgm

Extracts the threshold voltage from an I_d–V_{gs} curve using the maximum transconductance method. The threshold voltage is defined as the gate–voltage axis intercept of the tangent line at the maximum transconductance g_m point. The gate bias at which the maximum transconductance occurs is computed using parabolic interpolation. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractVtgm -out <var_name> -v <list_of_r> -i <list_of_r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the threshold voltage. (Real number, no default)
`-v <list_of_r>`	List containing the gate voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the drain current values. (List of real numbers, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Vtgm"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3f"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Extract Vtgm for a p-MOSFET
load_file IdVg_des.plt -name DC

set Vgs [get_variable_data "gate OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]
ext::AbsList -out absIds -x $Ids

ext::ExtractVtgm -out Vtgm -name "out" -v $Vgs -i $absIds
puts "Vt (Max gm method) is [format %.3f $Vtgm] V"
#-> Vt (Max gm method) is -0.234 V
```

ext::ExtractVti

Extracts the threshold voltage for a given subthreshold current level from an I_d–V_{gs} curve. The threshold voltage is defined as the gate voltage at which the drain current reaches the current level. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractVti -out <var_name> -v <list_of_r> -i <list_of_r> -io <r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the threshold voltage. (Real number, no default)
`-v <list_of_r>`	List containing the gate voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the drain current values. (List of real numbers, no default)
`-io <r>`	Subthreshold current level. (Real number, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Vti"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3f"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVg_des.plt -name DC

set Vgs [get_variable_data "gate OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]

ext::ExtractVti -out Vti -name "out" -v $Vgs -i $Ids -io 1e-7
puts "Vti (Vg at Io= 1e-7 A/um) is [format %.3f $Vti] V"
#-> Vti (Vg at Io= 1e-7 A/um) is 0.282 V
```

ext::ExtractVtsat

Extracts the threshold voltage from a $\sqrt{I_d} - V_{gs}$ curve. The threshold voltage is defined as the intercept with the gate-voltage axis from the point of maximum slope of the $\sqrt{I_d} - V_{gs}$ curve. The gate bias at which the maximum slope of the $\sqrt{I_d} - V_{gs}$ curve occurs is computed using parabolic interpolation. The curve is represented by two Tcl lists: one contains the voltage points and one contains the corresponding current values.

Syntax

```
ext::ExtractVtsat -out <var_name> -v <list_of_r> -i <list_of_r>
   [-name <string>] [-f <string>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the value of the threshold voltage. (Real number, no default)
`-v <list_of_r>`	List containing the gate voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the drain current values. (List of real numbers, no default)
`-name <string>`	Name of the extracted variable to appear in the Sentaurus Workbench Family Tree. **NOTE** If `-name "noprint"` is used, Sentaurus Workbench extraction is suppressed. **NOTE** If `-name "out"` is used, the name of the variable specified by the `-out` keyword also is used as the name that appears in the Sentaurus Workbench Family Tree. (String, default: `"Vtsat"`)
`-f <string>`	Format string used to write the extracted variable to the Sentaurus Workbench Family Tree. (String, default: `"%.3f"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVg_des.plt -name DC
```

```
set Vgs [get_variable_data "gate OuterVoltage" -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]
ext::AbsList -out absIds -x $Ids
```

```
ext::ExtractVtsat -out Vtsat -name "out" -v $Vgs -i $absIds
puts "Vtsat is [format %.3f $Vtsat] V"
#-> Vtsat is 0.193 V
```

ext::FilterTable

Processes data from the Sentaurus Workbench Family Tree for the purpose of creating a graph of one Sentaurus Workbench parameter (y-values) as a function of another Sentaurus Workbench parameter (x-values) for a certain subset of experiments. The data is specified in the form of two lists identifying the x- and y-values, which are preprocessed to create a graph. The condition that an experiment must fulfill to be included in the graph is specified using a pair of target values and a corresponding list of Sentaurus Workbench parameters.

Syntax

```
ext::FilterTable -out <array_name> -x <list_of_r> -y <list_of_r>
   -conditions <array_name> -ncond <i> [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements X and Y. The values of the X element are a subset of a list of values, specified using the keyword `-x`. These values are in ascending order. The values of the Y element are a subset of a list of values, specified using the keyword `-y`. All entries of the 'y'-list that contain a nonnumeric value are ignored. (Array name, no default)
`-x <list_of_r>`	List containing the values of a Sentaurus Workbench parameter to be preprocessed: the x-values. (List of real numbers, no default)
`-y <list_of_r>`	List containing the values of a Sentaurus Workbench parameter to be preprocessed: the y-values. (List of real numbers, no default)
`-conditions <array_name>`	Array with two indices. The string-indexed array contains the elements `"Target"` and `"Values"`. The value of the `"Target"` element contains the required value of a Sentaurus Workbench parameter to be used as a filter condition. The `"Values"` element contains the corresponding value list of the Sentaurus Workbench parameter for all the experiments. The second integer counter enumerates the conditions. The enumerations start with 1. (Array name, no default)
`-ncond <i>`	Number of conditions contained in the array specified using the keyword `-conditions`. (Integer, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Plot Vt roll-off curve for PMOS under stress
set Types  [list nMOS  nMOS  nMOS  nMOS  nMOS  nMOS  nMOS  nMOS \
                 pMOS  pMOS  pMOS  pMOS  pMOS  pMOS  pMOS  pMOS]
set Lgs    [list 0.090 0.090 0.045 0.045 0.130 0.130 0.065 0.065 \
                 0.065 0.065 0.045 0.045 0.130 0.130 0.090 0.090]
set Stress [list no    yes    no    yes    no    yes    no    yes \
                 no    yes    no    yes    no    yes    no    yes]
set Vtgms [list 0.424 0.0374 0.313 0.263 0.414 0.364  0.408 0.358 \
               -0.344 -0.294 -0.232 -0.182 x    x    -0.374 -0.324]

set Conditions(Target,1) "pMOS"
set Conditions(Values,1) $Types
set Conditions(Target,2) "yes"
set Conditions(Values,2) $Stress
ext::FilterTable -out LgVt -x $Lgs -y $Vtgms -conditions Conditions -ncond 2

create_variable -name Lg -dataset VtgmLg -values $LgVt(X)
create_variable -name Vtgm -dataset VtgmLg -values $LgVt(Y)
create_plot -1d -name Plot_VtRollOff
create_curve -name VtRollOff -dataset VtgmLg -axisX "Lg" -axisY "Vtgm"

puts "Lg= $LgVt(X)"
puts "Vtgm= $LgVt(Y)"
#-> Lg= 0.045 0.065 0.090
#-> Vtgm= -0.182 -0.294 -0.324
```

Usage Under Sentaurus Workbench

In a Sentaurus Visual 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. For example, the lists `Types`, `Lgs`, `Stress`, and `Vtgms` in the above example are generated automatically as a result of the following commands in the Sentaurus Visual script:

```
set Types  [list @Type:all@]
set Lgs    [list @lgate:all@]
set Stress [list @stress:all@]
set Vtgms  [list @Vt:all@]
```

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

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. For example, the Tcl list `Vtgms` contains the extracted values for the threshold voltage for each respective experiment.

> **NOTE** The values in the various lists may or may not be numeric, and the values may not necessarily be ordered.

The lists of x- and y-values, which will be processed (filtered) to create the graph, are specified using the keywords `-x` and `-y` in the procedure `ext::FilterTable`. In the above example, the lists of gate lengths (`-x $Lgs`) and V_{tgm} values (`-y $Vtgms`) are processed by `ext::FilterTable`.

The keyword `-conditions` controls the conditions an experiment must fulfill to be included in the graph. The total number of conditions is specified by the keyword `-ncond`. All the conditions are specified in a string-indexed array using the keyword `-conditions`. Each condition is defined by both a target value and a corresponding list of Sentaurus Workbench parameters. The target value is the required value of the parameter to be used as a filter condition.

Each element of the string-indexed array has two indices. The first index is either `"Target"` or `"Values"`. The second index is the condition number. For each condition number:

- The target value is specified using the `"Target"` element (element with first index named `"Target"`) of the array.
- The corresponding list of Sentaurus Workbench parameters is specified using the `"Values"` element (element with first index named `"Values"`) of the array.

In the above example, the following code filters out the gate length (L_g) and threshold voltage (V_{tgm}) values for PMOS devices (condition number 1). This condition is defined using the array named `Conditions`:

```
set Conditions(Target,1) "pMOS"
set Conditions(Values,1) $Types
```

Here, the target value is `"pMOS"` and the corresponding list of Sentaurus Workbench parameters is `Types`.

To filter out L_g and V_{tgm} values for devices under stress (condition number 2), the following additional elements of the `Conditions` array are defined:

```
set Conditions(Target,2) "yes"
set Conditions(Values,2) $Stress
```

As a result of specifying both the conditions (`-conditions Conditions -ncond 2`), the procedure `ext::FilterTable` filters out L_g and V_{tgm} values for PMOS devices under stress.

In the above example, if both the conditions are defined in the `Conditions` array but the number of conditions is set to 1 (`-conditions Conditions -ncond 1`), the procedure filters out the gate length and V_{tgm} values for all the PMOS devices (with and without stress). The second condition will not be taken into account.

The procedure returns an array (specified by the keyword `-out`) with a one string-valued index. The index contains the elements X and Y. The values of the X element are a subset of a list of values, specified using the keyword `-x`. These values are in ascending order. The values of the Y element are a subset of a list of values, specified using the keyword `-y`. These lists in the array can be used to create a graph.

In the above example, the procedure returns the array `LgVt` (`-out LgVt`) consisting of a list of L_g values and a list of V_{tgm} values for PMOS devices under stress. These lists can be used directly to create the V_t roll-off curve:

```
create_variable -name Lg -dataset VtgmLg -values $LgVt(X)
create_variable -name Vtgm -dataset VtgmLg -values $LgVt(Y)
create_plot -1d -name Plot_VtRollOff
create_curve -name VtRollOff -dataset VtgmLg -axisX "Lg" -axisY "Vtgm"
```

As an additional feature, the `ext::FilterTable` procedure ignores all entries of the y-values that contain a nonnumeric value. Use this feature to omit failed extractions. In the tool input file that performs the extraction, for example, a previous Sentaurus Visual tool instance, use the `#set` directive to preset the extracted variable to the value `x`:

```
#set Vtgm x
...
ext::ExtractVtgm -out Vtgm -name "out" -v $Vgs -i $absIds
```

The actual extraction process, here using the `ext::ExtractVtgm` procedure, overwrites the preset value `x` with the actual value. However, if the extraction process fails, the preset value persists.

The output of the above example shows that the V_{tgm} value (`= x`) for the 130 nm gate length (`Lg=0.130`) PMOS device under stress is not included in the array `LgVt`. In addition, the gate lengths in the array `LgVt` are in ascending order:

```
puts "Lg= $LgVt(X)"
puts "Vtgm= $LgVt(Y)"
#-> Lg= 0.045 0.065 0.090
#-> Vtgm= -0.182 -0.294 -0.324
```

ext::FindExtrema

Computes all local extrema (either maxima or minima) of a curve. The curve is represented by two Tcl lists, one containing the x-values (independent variable) and one containing the corresponding y-values (dependent variable).

NOTE If a curve exhibits a flat top or bottom (two or more neighboring x-values have the same y-value), then the last x-value is returned as the extrema point.

Syntax

```
ext::FindExtrema -out <array_name> -x <list_of_r> -y <list_of_r>
   [-type "max" | "min"] [-eps <r>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements X and Y. The values of the X element are the x-values corresponding to all the extrema. The values of the Y element are the extrema. (Array name, no default)
`-x <list_of_r>`	List containing the x-values. These must be in either ascending or descending order. (List of real numbers, no default)
`-y <list_of_r>`	List containing the y-values. (List of real numbers, no default)
`-type "max" | "min"`	Selects whether to extract the maxima (`"max"`) or minima (`"min"`) of a curve. Default: `"max"`
`-eps <r>`	If the difference between two adjacent elements of the list specified using the keyword `-y` is less than the value of `-eps`, both elements are considered to be equal. (Real number, default: 1×10^{-10})
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None

Example

```
set Xs [list 0 1.5 2.0 3.0]
set Ys [list 0 1.0 0.5 2.0]
ext::FindExtrema -out XY -x $Xs -y $Ys
puts "All maxima = $XY(Y)"
puts "All x values corresponding to the maxima = $XY(X)"
#-> All maxima = 1.0 2.0
#-> All x values corresponding to the maxima = 1.5 3.0
```

ext::FindVals

For a given target x-value, this procedure extracts all of the corresponding interpolated y-values in a curve. The curve is represented by two Tcl lists: one contains the x-values and one contains the corresponding y-values.

NOTE To find the interpolated x-values for a given y-value, swap the value of the keywords `-x` and `-y`.

Syntax

```
ext::FindVals -out <list_name> -x <list_of_r> -y <list_of_r> -xo <r>
   [-yLog 0 | 1] [-xLog 0 | 1] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <list_name>`	Name of a list to store the y-values. (List name, no default)
`-x <list_of_r>`	List containing the x-values. (List of real numbers, no default)
`-y <list_of_r>`	List containing the y-values. (List of real numbers, no default)
`-xo <r>`	Target x-value. (Real number, no default)
`-yLog 0 | 1`	Selects linear (`0`) or logarithmic (`1`) interpolation for y-axis values. Default: `0`
`-xLog 0 | 1`	Selects linear (`0`) or logarithmic (`1`) interpolation for x-axis values. Default: `0`
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Xs [list 0.0 1.0 2.0 3.0 4.0]
set Ys [list 0.0 2.0 4.0 2.0 0.0]

# Find all the elements of Xs corresponding to Ys= 2.0
ext::FindVals -out xos -x $Ys -y $Xs -xo 2.0
puts "The elements of Xs corresponding to Ys= 2.0 are $xos"
```

```
# Find the first element of Xs corresponding to Ys= 2.0
ext::ExtractValue -out xos -name "noprint" -x $Ys -y $Xs -xo 2.0
puts "The first element of Xs corresponding to Ys= 2.0 is $xos"
#-> The elements of Xs corresponding to Ys= 2.0 are 1.0 3.0
#-> The first element of Xs corresponding to Ys= 2.0 is 1.0
```

ext::LinFit

Performs a linear fit $y = x \bullet m + b$ to a curve using least-squares regression. The curve is represented by two Tcl lists, one containing the x-values (independent variable) and one containing the corresponding y-values (dependent variable).

Syntax

```
ext::LinFit -out <array_name> -x <list_of_r> -y <list_of_r>
   -xmin <r> -xmax <r> [-npar 1 | 2] [-weighted "off" | "on"]
   [-weights <list_of_r>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements X, Y, Yestimate, residuals, slope, yintercept, n, dof, and RMSE. The values of the X element are a subset of a list of values, specified using the keyword `-x`. The values of the Y element are a subset of a list of values, specified using the keyword `-y`. The values of the Yestimate, residuals, slope, yintercept (for `-npar 2`), dof, and RMSE elements are the estimated Y values ($\hat{y}_i$), the residuals (e_i), the estimated slope ($\hat{m}$), the estimated y-intercept ($\hat{b}$), the degrees of freedom (dof), and the root-mean-square error (RMSE), respectively. The value of the n element is the number of elements (n) in the list represented by the X element. (Array name, no default)
`-x <list_of_r>`	List containing the x-values. These must be in either ascending or descending order. (List of real numbers, no default)
`-y <list_of_r>`	List containing the y-values. (List of real numbers, no default)
`-xmin <r>`	Minimum x-value in the range of x-values over which the linear fit is performed. (Real number, no default)
`-xmax <r>`	Maximum x-value in the range of x-values over which the linear fit is performed. (Real number, no default)
`-npar 1 | 2`	Number of computed parameters. If `-npar 1` is used, only the slope is computed and the y-intercept is assumed to be 0. Default: `2`
`-weighted "off" | "on"`	Selects either unweighted (`"off"`) or weighted (`"on"`) linear regression. Default: `"off"`
`-weights <list_of_r>`	List containing the values of the weights for each x-value. (List of real numbers, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None

Example

```
set Xs [list 20 60 100 140 180 220 260 300 340 380]
set Ys [list 0.18 0.37 0.35 0.78 0.56 0.75 1.18 1.36 1.17 1.65]
ext::LinFit -out XY -x $Xs -y $Ys -xmin [lindex $Xs 0] -xmax [lindex $Xs end]
puts "Estimated slope= $XY(slope)"
puts "Estimated y-intercept= $XY(yintercept)"
puts "Root-MSE= $XY(RMSE)"
#-> Estimated slope= 0.00382
#-> Estimated y-intercept= 0.06924
#-> Root-MSE= 0.159
```

Linear Fitting Using Least-Squares Regression

The regression curve of Y as a function of X is:

$$y = x \bullet m + b \tag{8}$$

where m is the slope and b is the y-intercept.

The residual e_i of the i-th data point (x_i, y_i) is defined as:

$$e_i = y_i - \hat{y}_i \tag{9}$$

where $\hat{y}_i$ is the estimate of the y-value of the i-th data point.

The sum of squares due to error (SSE) is defined as:

$$\text{SSE} = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{10}$$

Here, the number of the data point is n.

The fitted or estimated regression line:

$$\hat{y} = x \bullet \hat{m} + \hat{b} \tag{11}$$

is computed by minimizing SSE. Here, $\hat{y}$, $\hat{m}$, and $\hat{b}$ are the estimated y-value, slope, and y-intercept, respectively [1].

The RMSE is defined as:

$$\mathrm{RMSE} = \sqrt{\frac{\mathrm{SSE}}{\mathrm{dof}}} \tag{12}$$

where the degrees of freedom (dof) for n data points is defined as:

$$\mathrm{dof} = n - 2 \tag{13}$$

ext::Linspace

Creates a list of *n* linearly spaced values between and including two real numbers (`xmin` and `xmax`).

Syntax

```
ext::Linspace -out <list_name> -xmin <r> -xmax <r> -n <i>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <list_name>`	Name of a list to store the list of linearly spaced values. (List name, no default)
`-xmin <r>`	Minimum x-value in the range of x-values over which the list is obtained. (Real number, no default)
`-xmax <r>`	Maximum x-value in the range of x-values over which the list is obtained. (Real number, no default)
`-n <i>`	Number of values created, where the value of `-n` should be a positive integer greater than 1. (Integer, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
ext::Linspace -out X -xmin 0 -xmax 1 -n 11
puts "Xs= $X"
==> Xs= 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
```

ext::LinTransList

Applies a linear transformation to the elements of a list. The elements of the list are replaced by the transformed values given by:

$$X' = X \bullet m + b \tag{14}$$

Syntax

```
ext::LinTransList -out <list_name> -x <list_of_r> [-m <r>] [-b <r>]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <list_name>`	Name of a list to store the list of transformed values. (List name, no default)
`-x <list_of_r>`	Input list. (List of real numbers, no default)
`-m <r>`	Slope of the linear transformation. (Real number, default: `1.0`)
`-b <r>`	Offset of the linear transformation. (Real number, default: `0.0`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file IdVg_des.plt -name DC
create_plot -1d -name Plot_IdVg
set Vgs [get_variable_data "gate OuterVoltage"  -dataset DC]
set Ids [get_variable_data "drain TotalCurrent" -dataset DC]

ext::LinTransList -out VgTrans -x $Vgs -b 0.55 ;# Shift Vg values by 0.55 V
ext::LinTransList -out IdTrans -x $Ids -m 1e6
                                       ;# Scale Id values from A/um to mA/mm

# Create the shifted and scaled Id-Vg curve
create_variable -name VgTrans -dataset IdVgTrans -values $VgTrans
create_variable -name IdTrans -dataset IdVgTrans -values $IdTrans
create_curve -name IdVgTrans -dataset IdVgTrans \
   -axisX "VgTrans" -axisY "IdTrans
```

ext::Log10List

Applies the log10 function to the elements of a list. The elements of the list are replaced by the function values.

Syntax

```
ext::Log10List -out <list_name> -x <list_of_r>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <list_name>`	Name of a list to store the list of values. (List name, no default)
`-x <list_of_r>`	Input list. (List of real numbers, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
ext::Log10List -out ys -x [list 10 100 1000]
puts "log10(x)= $ys"
==> log10(x)= 1.0 2.0 3.0
```

ext::RemoveDuplicates

For a pair of lists `x` and `y`, removes duplicate elements of the list `x` and the corresponding elements of the list `y`.

Syntax

```
ext::RemoveDuplicates -out <array_name> -x <list_of_r> -y <list_of_r>
   [-eps <r>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements X and Y. The values of the X element are a subset of a list of values, specified using the keyword `-x`. These do not contain duplicate values. The corresponding elements of the list specified using the keyword `-y` are stored in the Y element. The values of the Y element are a subset of a list of values, specified using the keyword `-y`. (Array name, no default)
`-x <list_of_r>`	Input list. (List of real numbers, no default)
`-y <list_of_r>`	Input list. (List of real numbers, no default)
`-eps <r>`	If the difference between two adjacent elements of the list specified using the keyword `-x` is less than `-eps`, the first element is removed. (Real number, default: 10^{-40})
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set x [list 1  1  2  3  1  1  2  2  2  3]
set y [list 10 20 30 40 50 60 70 80 90 100]
ext::RemoveDuplicates -out XY -x $x -y $y
set Xs $XY(X)
set Ys $XY(Y)
puts "Xs= $Xs"
puts "Ys= $Ys"
==> Xs= 1 2 3 1 2 3
==> Ys= 20 30 40 60 90 100
```

ext::RemoveZeros

For a pair of lists `x` and `y`, removes zero elements of the list `x` and the corresponding elements of the list `y`.

Syntax

```
ext::RemoveZeros -out <array_name> -x <list_of_r> -y <list_of_r>
   [-iplists "x" | "y" | "xy"] [-eps <r>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements X and Y. The values of the X element are a subset of a list of values, specified using the keyword `-x`. These do not contain zero values. The corresponding elements of the list specified using the keyword `-y` are stored in the Y element. The values of the Y element are a subset of a list of values, specified using the keyword `-y`. (Array name, no default)
`-x <list_of_r>`	Input list. (List of real numbers, no default)
`-y <list_of_r>`	Input list. (List of real numbers, no default)
`-iplists "x" | "y" | "xy"`	Input list from which zeros are removed. If `-iplists "x"` is used, the zeros are removed from the list specified using the keyword `-x`. If `-iplists "xy"` is used, zeros are removed from both lists specified using the keywords `-x` and `-y`. (String, no default)
`-eps <r>`	If an element of the list specified using the keyword `-x` is less than value of `-eps`, it is removed. (Real number, default: 10^{-40})
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set xs [list 0 1 2 3 0 -1 ]
set ys [list 10 20 30 40 50 0 ]
ext::RemoveZeros -out XY -x $xs -y $ys -iplists "x"
puts "Xs= $XY(X)"
puts "Ys= $XY(Y)"
==> Xs= 1 2 3 -1
==> Ys= 20 30 40 0
```

ext::SubLists

Creates a pair of sublists from a pair of lists. One of the lists is the list of x-values. The second list is the list of y-values. The sublist is created using a range of x-values.

NOTE To create a pair of sublists using a range of y-values, swap the keywords `-x` and `-y`, and specify the range of y-values using the keywords `-xmin` and `-xmax`. To create a sublist from a single list, specify the value of the list using both the `-x` and `-y` keywords.

Syntax

```
ext::SubLists -out <array_name> -x <list_of_r> -y <list_of_r>
   -xmin <r> -xmax <r> [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements X and Y. The values of the X element are a subset of a list of values, specified using the keyword `-x`. The values of the Y element are a subset of a list of values, specified using the keyword `-y`. (Array name, no default)
`-x <list_of_r>`	List containing the x-values. These must be in either ascending or descending order. (List of real numbers, no default)
`-y <list_of_r>`	List containing the y-values. (List of real numbers, no default)
`-xmin <r>`	Minimum x-value in the range of x-values over which the sublist is obtained. (Real number, no default)
`-xmax <r>`	Maximum x-value in the range of x-values over which the sublist is obtained. (Real number, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None

Example

```
set xs [list 1 2 3 4 5 6 7]
set ys [list 10 20 30 40 50 60 70]
ext::SubLists -out XY -x $xs -y $ys -xmin 2 -xmax 5
puts "Xs= $XY(X)"
puts "Ys= $XY(Y)"
==> Xs= 2 3 4 5
==> Ys= 20 30 40 50
```

lib::SetInfoDef

Sets the default information level.

NOTE Level 0: Warning, error, or status messages only.
Level 1: Echo results.
Level 2: Show progress and some debug information.
Level 3: Show all debug information.

The local info level also can be set using the `-info` keyword of the procedures in the extraction library.

Syntax

```
lib::SetInfoDef 0 | 1 | 2 | 3
```

Argument	Description
`<info_level>`	Sets the default info level. Default: 0

Returns

None.

Example

```
lib::SetInfoDef 2
```

References

[1] W. H. Press *et al.*, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge: Cambridge University Press, 2nd ed., 1992.

APPENDIX F Impedance Field Method Data Postprocessing Library

This appendix provides information about the impedance field method data postprocessing library.

Overview

The impedance field method in Sentaurus Device provides an accurate and efficient way to evaluate the effects of random variability on the electrical behavior of semiconductor devices (see Sentaurus™ Device User Guide, Chapter 23 on page 675).

Within the statistical impedance field method (sIFM), Sentaurus Device generates many randomized realizations of a reference device. For example, 10000 realizations with different randomized doping distributions, different randomized gate oxide thicknesses, different randomized metal grain boundaries, and so on.

For each of these individual randomizations, Sentaurus Device computes, at each bias point, the linear current response of the randomizations with respect to the reference device.

The impedance field method (IFM) data postprocessing library helps to manage and analyze large amounts of linear current response data. For example, the IFM library allows you to conveniently apply standard statistical analysis methods to data such as computing and visualizing the distribution and comparing it to a Gaussian distribution.

The IFM library also supports the construction of the individual electrical characteristics of the randomized devices from the electrical characteristics of the reference device and the linear current response data.

The IFM library is loaded automatically when Sentaurus Visual starts. However, if you have disabled the automatic loading of extension libraries, you can load the IFM library explicitly with the command:

```
load_library ifm
```

Syntax Conventions

The IFM library uses a unique namespace identifier (`ifm::`) for its procedures. All procedures and variables associated with this library are called with the namespace identifier prepended, for example:

```
ifm::<proc_name>
```

Each procedure has several arguments. The IFM library uses an input parser that accepts arguments of the form:

```
-keyword <value>
```

NOTE All Sentaurus Visual libraries support the standard Sentaurus Visual syntax in which keywords are preceded by a dash. For backward compatibility, all Sentaurus Visual libraries continue to support the `keyword= <value>` syntax as well. For each procedure call, you can use either the `-keyword <value>` syntax or the `keyword= <value>` syntax. However, within any one procedure call, only one type of syntax can be used. Only the new syntax is documented. If you want to continue using the `keyword= <value>` syntax, you also can insert whitespace between the keyword and the equal sign, for example, `keyword = <value>`. Omitting the whitespace between the equal sign and the value field will result in a failure if the value is a de-referenced Tcl variable. Use `keyword= $val` (*not* `keyword=$val`).

The parser accepts arguments in any order. For some arguments, default values are predefined. Such arguments may be omitted. If arguments for which no defaults are predefined are omitted, the procedure will exit with an error message. In addition, unrecognized arguments result in an error message.

Some procedures of the IFM library compute large and complex data structures. For such data structures, the standard Tcl method of using the return value of the procedure to pass results back to the calling program is not suitable. Therefore, for some datasets, the IFM library uses a *passing-by-reference* method to exchange information between the procedure and the calling program. Procedure arguments that use the passing-by-reference method are identified with `-keyword <var_name>`, `<list_name>`, or `<array_name>`.

The following conventions are used for the syntax of Tcl commands:

- Angle brackets – <> – indicate text that must be replaced, but they are *not* part of the syntax. In particular, the following type identifiers are used:
  - `<r>`: Replace with a real number, or a de-referenced Tcl variable that evaluates to a real number. For example: `$val`.

- `<i>`: Replace with an integer, or a de-referenced Tcl variable that evaluates to an integer. For example: `$i`.
- `<string>`: Replace with a string, or a de-referenced Tcl variable that evaluates to a string. For example: `$file`.
- `<list_of_r>`: Replace with a list of real numbers, or a de-referenced Tcl variable that evaluates to a list of real numbers. For example: `$values`.
- `<list_of_strings>`: Replace with a list of strings, or a de-referenced Tcl variable that evaluates to a list of strings. For example: `$files`.
- `<var_name>`: Replace with the *name* of a local Tcl variable.
  For example: `val` (*not* `$val`).
- `<list_name>`: Replace with the *name* of a local Tcl list.
  For example: `values` (*not* `$values`).
- `<array_name>`: Replace with the *name* of a local Tcl array.
  For example: `myarray` (*not* `$myarray`).

- Brackets – [] – indicate that the argument is optional, but they are *not* part of the syntax.
- A vertical bar – | – indicates options, only one of which can be specified.

Help for Procedures

To request help on a specific procedure, set the `-help` keyword to `1`:

```
ifm::<proc_name> -help 1
```

If this command is included in a Sentaurus Visual file, when Sentaurus Visual is executed in:

- Batch mode in Sentaurus Workbench, the help information is printed to the runtime output file (with the extension `.out`) of the corresponding Sentaurus Visual node.
- Interactive mode in Sentaurus Workbench, the help information is displayed in the Tcl Command panel as well as printed in the Sentaurus Visual output file.

You also can type this command in the Tcl Command panel of the graphical user interface, in which case, the help information is displayed in the same panel.

Output of Procedures

All procedures of the IFM library pass the results back to the calling program by storing the results in a Tcl variable or a Tcl array. The name of this Tcl variable or array is specified as the value of the `-out` keyword.

If there are errors in the IFM library procedures, the behavior of Sentaurus Visual depends on whether it is executed in batch mode or interactive mode in Sentaurus Workbench. In batch mode, Sentaurus Visual exits and an error message is printed only in the Sentaurus Visual error file (with the extension `.err`). In interactive mode, the error message is displayed in the Tcl Command panel as well as printed in the Sentaurus Visual error file.

All procedures also print several messages (including warning messages). If Sentaurus Visual is executed in batch mode, the messages are printed only in the Sentaurus Visual output file; whereas, in interactive mode, the messages are displayed in the Tcl Command panel as well as printed in the Sentaurus Visual output file.

The amount of information printed depends on the information level specified by the procedure `lib::SetInfoDef`.

ifm::Gauss

Computes the y-value of a normalized Gaussian distribution for a given x-value:

$$y = \frac{N}{\sigma\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \tag{15}$$

where N is the norm of the Gaussian distribution, μ is the average, and σ is the standard deviation.

Syntax

```
ifm::Gauss -out <var_name> -x <r> -moments <array_name> [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the corresponding y-value of the normalized Gaussian distribution.
`-x <r>`	The x-value. (Real number, no default)
`-moments <array_name>`	Name of an array with one string-valued index, which contains the elements `norm`, `ave`, and `std_dev`. The values of these elements contain the requested norm, the average, and the standard deviation of the Gaussian. (Array name, no default)
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Moments(norm)    1.0
set Moments(ave)     0.0
set Moments(std_dev) 1.0
ifm::Gauss -out G -x 0.1 -moments Moments
puts "The result is $G"
```

ifm::GetDataQuantiles

Computes quantiles for a list of random variables.

This procedure sorts a list of random values and associates each value with the corresponding quantile (a value between 0 and 1).

Syntax

```
ifm::GetDataQuantiles -out <array_name> -rvs <list_of_r> [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index, which contains the elements `X` and `Y`. The values of these elements contain the sorted random variables (`X`) and the corresponding quantiles (`Y`). (Array name, no default)
`-rvs <list_of_r>`	List of random variables. (List of real numbers, no default)
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set RanVals [list -1.657 0.7661 2.142 1.189 -1.919 -0.6670 -0.1915 0.3662]
ifm::GetDataQuantiles -out DataQ -rvs $RanVals
puts $DataQ(X)
#-> -1.919 -1.657 -0.6670 -0.1915 0.3662 0.7661 1.189 2.142
puts $DataQ(Y)
#-> 0.0625 0.1875 0.3125 0.4375 0.5625 0.6875 0.8125 0.9375
```

ifm::GetGaussian

Computes either a Gaussian curve:

$$G(x) = \frac{N}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \tag{16}$$

or the quantiles of a Gaussian curve:

$$Q(x) = \int_{xmin}^{x} G(x)dx \tag{17}$$

where N is the norm of the Gaussian distribution, μ is the average, and σ is the standard deviation.

Syntax

```
ifm::GetGaussian -out <array_name> -moments <array_name>
   [-nsam <i>] [-type f | q] [-xmin <r>] [-xmax <r>] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index, which contains the elements `X` and `Y`. The values of these elements represent the lists of x- and y-values of the Gaussian or quantile. (Array name, no default)
`-moments <array_name>`	Name of an array with one string-valued index, which contains the elements `norm`, `ave`, and `std_dev`. The values of these elements contain the requested norm, the average, and the standard deviation of the Gaussian. (Array name, no default)
`-nsam <i>`	Number of sample points. (Integer, default: `80`)
`-type f | q`	`f`: A Gaussian curve defined by the given moments is returned. `q`: The quantiles of this Gaussian are returned. Default: `f`
`-xmin <r>`	Starting x-value. (Real number, default: `-3.0`)
`-xmax <r>`	Ending x-value. (Real number, default: `3.0`)
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
create_plot -1d -name Gaussian
select_plots Gaussian
set Moments(norm)    1.0
set Moments(ave)     0.0
set Moments(std_dev) 1.0
ifm::GetGaussian -out Gaussian -type f -moments Moments \
   -nsam 100 -xmin -3.5 -xmax 3.5
create_variable -name "GX" -dataset GXY -values $Gaussian(X)
create_variable -name "GY" -dataset GXY -values $Gaussian(Y)
create_curve -name gauss -dataset GXY -axisX "GX" -axisY "GY"
```

ifm::GetHistogram

Computes x- and y-lists to be used to plot a histogram for a given list of random variables, a given plotting range, and a given number of bins.

Syntax

```
ifm::GetHistogram -out <array_name> -rvs <list_of_r> -xmin <r> -xmax <r>
   [-nbin <i>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index, which contains the elements `X` and `Y`. The values of these elements represent the lists of x- and y-values of the histogram. (Array name, no default)
`-rvs <list_of_r>`	List of random variables. (List of real numbers, no default)
`-xmin <r>`	Starting x-value of histogram. (Real number, no default)
`-xmax <r>`	Ending x-value of histogram. (Real number, no default)
`-nbin <i>`	Number of bins for the histogram. (Integer, default: `40`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
create_plot -1d -name Histogram
select_plots Histogram
set Rxs [list 1 1.1 2 5 5.1 5.3 6]
ifm::GetHistogram -out Histogram -rvs $Rxs -xmin 0 -xmax 6 -nbin 6
create_variable -name "X" -dataset XY -values $Histogram(X)
create_variable -name "Y" -dataset XY -values $Histogram(Y)
create_curve -name his -dataset XY -axisX "X" -axisY "Y"
```

ifm::GetMoments

Computes the norm, the average, the root mean square (rms), the standard deviation, the skewness, and the excess kurtosis for a given list of random variables:

$$\mu = \frac{1}{N}\sum_{i}^{N} x_i \tag{18}$$

$$x_{\mathrm{rms}} = \sqrt{\frac{1}{N}\sum_{i}^{N} x_i^2} \tag{19}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i}^{N} (x_i - \mu)^2} \tag{20}$$

$$y = \frac{1}{N\sigma^3}\sum_{i}^{N} (x_i - \mu)^3 \tag{21}$$

$$k = \frac{1}{N\sigma^4}\sum_{i}^{N} (x_i - \mu)^4 - 3 \tag{22}$$

where the norm N is given by the number of random values, the index i enumerates the random values, μ is the average, x_{rms} is the root mean square, σ is the standard deviation, y is the skewness, and k is the excess kurtosis.

Syntax

```
ifm::GetMoments -out <array_name> -rvs <list_of_r> [-info 0 | 1 | 2 | 3]
   [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of a string-valued array to store the computed moments. This array contains the elements `norm`, `ave`, `rms`, `std_dev`, `skew`, and `kurtx`. The values of these elements contain the computed norm, the average, the rms, the standard deviation, the skewness, and the excess kurtosis of the list of random variables. (Array name, no default)
`-rvs <list_of_r>`	List of random variables. (List of real numbers, no default)

`-info 0	1	2	3`	Sets local info level. Default: `0`
`-help 0	1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Vs [list 1 2 3 4 5]
ifm::GetMoments -out Moments -rvs $Vs
puts $Moments(norm)
#->5
puts $Moments(ave)
#->3
puts $Moments(rms)
#->3.31662479036
puts $Moments(std_dev)
#->1.41421356237
puts $Moments(skew)
#->0.0
puts $Moments(kurtx)
#->-1.3
```

ifm::GetMOSIVs

Constructs the randomized I_d–V_g curves for MOS-type devices for one or more randomization sources.

The boundary condition that links the linear current response $\delta I_{\nu,d}$ to the nodal drain current $dI_{\nu,d}$ and the gate voltage $dV_{\nu,g}$ variations is given by:

$$dI_{\nu,d} = \delta I_{\nu,d} + y_{d,g} dV_{\nu,g} \tag{23}$$

The index ν enumerates the randomizations. The given equation gives the freedom to interpret the linear current response directly as a change of the drain current:

$$dI_{\nu,d} = \delta I_{\nu,d} \tag{24}$$

Alternatively, you can interpret it as an adjustment of the gate bias:

$$dV_{\nu,g} = -\frac{\delta I_{\nu,d}}{y_{d,g}} \tag{25}$$

For the linearized system, the following two methods yield identical results:

- The gate voltage adjustment method (`dV`):

$$I_{\nu,d} = I_{\mathrm{ref},d}(V_{\mathrm{ref},g} - dV_{\nu,g}) \tag{26}$$

- The drain current adjustment method (`dI`):

$$I_{\nu,d} = I_{\mathrm{ref},d}(V_{\mathrm{ref},g}) + dI_{\nu,d} \tag{27}$$

The drain current and the gate voltage of the reference device are given by $I_{\mathrm{ref},d}$ and $V_{\mathrm{ref},g}$, respectively.

The equivalence of these two methods can be verified by expanding the two equations into a Taylor series. For a nonlinear system, the two formulations are not equivalent and, depending on the details of the nonlinearity, one or the other method may give more accurate results. To better understand the implications, consider two limiting cases:

(i) Steeply rising I_d–V_g in the subthreshold and near-threshold regime
In this regime, variability effects are well approximated by a threshold voltage shift. This means that, while both $\delta I_{\nu,d}$ and $y_{d,g}$ increase exponentially with increasing gate bias, the ratio of the two quantities $dV_{\nu,g}$ remains approximately constant. While large values of $dI_{\nu,d}$ can result in unphysical negative output currents for some randomizations, when using the drain current adjustment method, the gate voltage adjustment method always guarantees positive and physical output currents.

(ii) Saturating I_d–V_g at low drain bias and high gate bias
In this regime, the transconductance $y_{d,g}$ vanishes and, therefore, $dV_{\nu,g}$ diverges, while $\delta I_{\nu,d}$ remains approximately constant. Consequently, the large values of $dV_{\nu,g}$ can result in unphysical gate voltages (non-monotonous, or less than ground, or larger than the supply voltage) for some randomizations, when using the gate voltage adjustment method. The drain current adjustment method, however, always guarantees monotonous and physical input voltages.

For the I_d–V_g characteristic, the transition point between the subthreshold and near-threshold regime and the saturation regime can be defined as the point of maximal transconductance and, therefore, you can apply either the gate voltage adjustment method or the drain current adjustment method, depending on the sign of the derivative of the transconductance. This observation is the foundation of the third method:

- The weighted method (`weighted`):

$$I_{\nu,d} = I_{\mathrm{ref},d}\left(V_{\mathrm{ref},g} - \frac{1+W}{2}dV_{\nu,g}\right) \qquad (28)$$
$$I_{\nu,d} = I_{\mathrm{ref},d}(V_{\mathrm{ref},g}) + \frac{1-W}{2}dI_{\nu,d}$$

The weights W are computed internally by calling ifm::GetMOSWeights on page 407.

The weighted method switches between the gate voltage adjustment method and the drain current adjustment method to avoid artificial over-adjustments of the currents or voltages. In situations with relatively large adjustments at the transition point, discontinuities and overlaps may be observed. The `smooth` option activates a smoothening procedure to eliminate these artifacts at the transition point.

An alternative weighting scheme (`SSweighted`) switches from the `dV` method to the `dI` method when the subthreshold slope becomes larger than the user-defined threshold `ss`. This method uses an error function to smooth out the transition, and you can control the smoothness by setting the normalization parameter `dss`. The weights are computed internally by calling ifm::GetMOSWeights on page 407.

For I_d–V_g sweeps in the linear regime, it is recommended to use one of the weighted methods, preferably the one that gives the smoothest resulting I–V curves.

Finally, the conceptually simpler exponential method ensures nonnegative currents and also avoids gate bias overshoots. This method often gives satisfactory results, but it violates the linearity assumption:

- The exponential method (`exp`):

$$I_{\nu,d} = I_{\mathrm{ref},d}\exp\left(\frac{dI_{\nu,d}}{I_{\mathrm{ref},d}}\right) \qquad (29)$$

Even when using the most appropriate I_d–V_g construction method, it may happen that for a certain randomization, the linear current response $dI_{v,d}$ becomes too large compared to the nominal current $I_{v,d}$ and the resulting I_d–V_g may exhibit an unexpected shape. This can result in the unreliable extraction of electrical parameters such as the subthreshold slope or the threshold voltage for these specific randomizations. You can flag and filter out such curves by looking at the maximum deviation $|dI_{v,d}/I_{v,d}|$. The maximum deviation is computed within a user-specified bias range for each constructed I_d–V_g and is accessible using the array element `maxdev` of the array that also contains the x- and y-values of the I_d–V_g curves. You can limit the bias range by specifying the `-vmin` and `-vmax` arguments.

Syntax

```
ifm::GetMOSIVs -out <array_name> -sifm <array_name> -nrow <var_name>
   -ncol <var_name> -v <list_of_r> -i <list_of_r> -y <list_of_r>
   -vmin <r> -vmax <r> -id <string>
   [-type IdVg] [-method dV | dI | weighted | SSweighted | exp]
   [-smooth yes | no] [-sgn 1 | -1] [-ss <r> -dss <r>]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of a two-indexed array to store the constructed I–V curves. The first index is string valued. The elements are X and Y. The second index is integer valued. It represents the randomization index. The array element values contain the x- and y-value lists of the respective I–V curves. The array element `maxdev` contains the maximum deviation: `maxdev` = max|*dI*/*I*| for `vmin` < V < `vmax` for the given I–V curve. It can be used to filter out curves for which `maxdev` is too big, for example, greater than 1. (Array name, no default)
`-sifm <array_name>`	Name of an array that contains the sIFM data. The array has three indices: The first index is string valued. The elements are the variability source identifiers. The second index is integer valued. It represents the row or bias index. The third index is integer valued. It represents the column or randomization index. The array element values contain the sIFM linear current response. (Array name, no default)
`-nrow <var_name>`	Name of a variable containing the number of rows (bias points) in the sIFM data. (Variable name, no default)
`-ncol <var_name>`	Name of a variable containing the number of columns (randomizations) in the sIFM data. (Variable name, no default)
`-v <list_of_r>`	List containing the reference voltage values. (List of real numbers, no default)
`-i <list_of_r>`	List containing the reference current values. (List of real numbers, no default)

`-y <list_of_r>`	List containing the relevant reference Y-matrix values. (List of real numbers, no default)
`-vmin <r>`	Minimum bias for dI/I monitoring.
`-vmax <r>`	Maximum bias for dI/I monitoring.
`-id <string>`	ID of the sIFM variability source. (String, no default)
`-type IdVg`	Currently, only `IdVg` is supported. Default: `IdVg`
`-method dV | dI | weighted | SSweighted | exp`	Selects the I–V construction method. Default: `weighted`
`-smooth yes | no`	Activates I–V smoothening for the weighted method. Default: `no`
`-sgn 1 | -1`	Set to `1` for NMOS or `-1` for PMOS. Default: `1`
`-ss <r>`	Subthreshold slope inflection point in mV/decade. (Real, only needed for `SSweighted`)
`-dss <r>`	Width of transition region in mV/decade. (Real, only needed for `SSweighted`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set IDs [list rdf ift SUM]
set FILEs [list]
foreach ID $IDs {
   lappend FILEs mos_${ID}_I_ndrain.csv
}

load_file mos_circuit_des.plt -name Data(DC)
load_file mos_ac_des.plt      -name Data(AC)

set adgs [get_variable_data "a(ndrain,ngate)" -dataset Data(AC)]
set Vgs  [get_variable_data "v(ngate)"        -dataset Data(DC)]
set Ids  [get_variable_data "i(mos,ndrain)"   -dataset Data(DC)]

ifm::ReadsIFM -out sIFM -nrow Nrow -ncol Ncol -files $FILEs -ids $IDs

create_plot -1d -name RanIV
select_plots RanIV
```

```
set j 42
ifm::GetMOSIVs -out IV_rdf -sifm sIFM -nrow Nrow -ncol Ncol \
   -method "weighted" -sgn 1.0 -v $Vgs -i $Ids -y $adgs -id "rdf" -smooth yes
create_variable -name V -dataset RanIV(rdf,$j) -values $IV_rdf(X,$j)
create_variable -name I -dataset RanIV(rdf,$j) -values $IV_rdf(Y,$j)
create_curve -name IV_rdf($j) -dataset RanIV(rdf,$j) -axisX "V" -axisY "I"

ifm::GetMOSIVs -out IV_ift -sifm sIFM -nrow Nrow -ncol Ncol \
   -method "weighted" -sgn 1.0 -v $Vgs -i $Ids -y $adgs -id "ift" -smooth yes
create_variable -name V -dataset RanIV(ift,$j) -values $IV_ift(X,$j)
create_variable -name I -dataset RanIV(ift,$j) -values $IV_ift(Y,$j)
create_curve -name IV_ift($j) -dataset RanIV(ift,$j) -axisX "V" -axisY "I"

ifm::GetMOSIVs -out IV_SUM -sifm sIFM -nrow Nrow -ncol Ncol \
   -method "weighted" -sgn 1.0 -v $Vgs -i $Ids -y $adgs -id "SUM" -smooth yes
create_variable -name V -dataset RanIV(SUM,$j) -values $IV_SUM(X,$j)
create_variable -name I -dataset RanIV(SUM,$j) -values $IV_SUM(Y,$j)
create_curve -name IV_SUM($j) -dataset RanIV(SUM,$j) -axisX "V" -axisY "I"
```

ifm::GetMOSWeights

Computes the weights for the construction of randomized MOSFET I_d–V_g curves.

This procedure computes, for each bias point, a weight between 1 and –1 to indicate that the gate-voltage adjustment method or the drain-current adjustment method is to be used.

For `-type IdVg_weighted`, the weights are set to 1 or –1 depending on the sign of the derivative of the transconductance.

For `-type IdVg_SSweighted`, the weights are computed as:

$$W = \mathrm{erf}\left(\frac{ss - SS}{dss}\right) \tag{30}$$

where SS is the subthreshold slope, and ss is the user-defined switch-over threshold. The smoothness of the transition is set by the user-defined normalization parameter dss.

Syntax

```
ifm::GetMOSWeights -out <list_name> -y <list_of_r>
   [-type IdVg_weighted | IdVg_SSweighted] [-i <list_of_r> -ss <r> -dss <r>]
   [-help 0 | 1]
```

Argument	Description
`-out <list_name>`	Name of a list to store the computed weights. (List name, no default)
`-y <list_of_r>`	List containing the relevant reference Y-matrix values. (List of real numbers, no default)
`-type IdVg_weighted | IdVg_SSweighted`	Selects the method used for the computation of the weights. Default: `IdVg_weighted`
`-i <list_of_r>`	List containing the reference current values. (List of real numbers, only needed for `IdVg_SSweighted`)
`-ss <r>`	Subthreshold slope inflection point in mV/decade. (Real, only needed for `IdVg_SSweighted`)
`-dss <r>`	Width of transition region in mV/decade. (Real, only needed for `IdVg_SSweighted`)
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file mos_ac_des.plt -name Data(AC)
set adgs [get_variable_data a(ndrain,ngate) -dataset Data(AC)]
ifm::GetMOSWeights -out Ws -y $adgs
create_variable -name W -dataset Data(AC) -values $Ws
create_plot -1d -name Weights
select_plots Weights
create_curve -name W -dataset Data(AC) -axisX "v(ngate)" -axisY "W"
create_curve -name Y -dataset Data(AC) -axisX "v(ngate)" \
   -axisY2 "a(ndrain,ngate)"
```

ifm::GetNoiseStdDev

Computes the drain current $\sigma(I_d)$ and the gate voltage $\sigma(V_g)$ standard deviation from the drain current noise spectral density $S_{d,d}$ (see Sentaurus™ Device User Guide, Chapter 23 on page 675), and the gate-to-drain admittance $y_{d,g}$:

$$\sigma(I_d) = \sqrt{S_{d,d} \cdot 1\text{Hz}} \tag{31}$$

$$\sigma(V_g) = \frac{\sqrt{S_{d,d} \cdot 1\text{Hz}}}{y_{d,g}} \tag{32}$$

Syntax

```
ifm::GetNoiseStdDev -out <array_name> -s <list_of_r> -y <list_of_r>
   [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements `I` and `V`. The value of the `I` element is a list with the current standard deviations. The value of the `V` element is a list with the voltage standard deviations. (Array name, no default)
`-s <list_of_r>`	List containing the current noise spectral density values. (List of real numbers, no default)
`-y <list_of_r>`	List containing the relevant reference Y-matrix values. (List of real numbers, no default)
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file "mos_ac_des.plt" -name Data(AC)
set adgs         [get_variable_data a(ndrain,ngate) -dataset Data(AC)]
set S_Ids(noise) [get_variable_data S_I(ndrain)     -dataset Data(AC)]

ifm::GetNoiseStdDev -out sigIV -s $S_Ids(noise) -y $adgs
create_plot -1d -name Sig
create_variable -name sigId(noise) -dataset Data(AC) -values $sigIV(I)
create_curve -name sigId(noise) -dataset Data(AC) \
   -axisX "v(ngate)" -axisY "sigId(noise)"
create_variable -name sigVg(noise) -dataset Data(AC) -values $sigIV(V)
```

```
create_curve -name sigVg(noise) -plot Sig -dataset Data(AC) \
   -axisX "v(ngate)" -axisY2 "sigVg(noise)"
```

ifm::GetQQ

Compares the quantiles of a given data distribution with the quantiles of a Gaussian distribution.

For each value in the quantiles of the Gaussian distribution, the matching (interpolated) value of the data distributions is found. Then, the data x-values corresponding to this match are plotted against the normalized Gaussian x-values $(x-\mu)/\sigma$.

Syntax

```
ifm::GetQQ -out <array_name> -dq <array_name> -gq <array_name>
   -moments <array_name> [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index, which contains the elements `X` and `Y`. The values of these elements represent the lists of x- and y-values of a quantile–quantile comparison curve. (Array name, no default)
`-dq <array_name>`, `-gq <array_name>`	Name of arrays containing the quantiles of the given data distribution (`dq`) and the quantiles of a corresponding Gaussian distribution (`gq`). The arrays `dq` and `gq` each have one string-valued index, which contains the elements `X` and `Y`. The values of these elements represent the lists of x- and y-values of the quantiles. (Array name, no default)
`-moments <array_name>`	Name of an array with one string-valued index, which contains the elements `norm`, `ave`, and `std_dev`. The values of these elements contain the requested norm, the average, and the standard deviation of the Gaussian. (Array name, no default)
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set RanVals [list -1.657 0.7661 2.142 1.189 -1.919 -0.6670 -0.1915 0.3662]
ifm::GetDataQuantiles -out DataQ -rvs $RanVals
ifm::GetMoments -out Moments -rvs $RanVals
ifm::GetGaussian -out GaussianQ -type q -moments Moments \
   -nsam 40 -xmin -3.5 -xmax 3.5

ifm::GetQQ -out QQ -dq DataQ -gq GaussianQ -moments Moments
create_plot -1d -name QQplot
select_plots QQplot
create_variable -name "QQX" -dataset QQXY -values $QQ(X)
create_variable -name "QQY" -dataset QQXY -values $QQ(Y)
create_curve -name qq -dataset QQXY -axisX "QQX" -axisY "QQY"
```

ifm::GetsIFMStdDev

Computes the drain current and the gate voltage standard deviation from the sIFM linear current responses and the gate-to-drain admittance.

For each bias point in the sIFM data file, this procedure reads the linear current responses and calls `ifm::GetMoments` to compute the drain current standard deviation $\sigma(I_d)$. The gate voltage $\sigma(V_g)$ standard deviation is obtained by dividing the drain current standard deviation by the gate-to-drain admittance $y_{d,g}$.

This procedure also supports the computation of the drain-current and the gate-voltage standard deviation for contact resistance variability.

Syntax

```
ifm::GetsIFMStdDev -out <array_name> -sIFM <string> -y <list_of_r>
   [-rsig <r> -ydd <list_of_r> -i <list_of_r>] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements `I` and `V`. The value of the `I` element is a list with the current standard deviations. The value of the `V` element is a list with the voltage standard deviations. (Array name, no default)
`-sIFM <string>`	Name of a comma-separated value (CSV) file containing the sIFM linear current responses. If this string is set to `"crv"` the contact resistance variability is computed instead. (String, no default)
`-y <list_of_r>`	List containing the relevant reference Y-matrix values. (List of real numbers, no default)

`-rsig <r>`	Standard deviation of contact resistance variability in Ohm. Activated when sIFM is set to `"crv"`. (Real, only needed for contact resistance variability)
`-ydd <list_of_r>`	List containing the reference Y(d,d) matrix elements. Activated when sIFM is set to `"crv"`. (List of real numbers, only needed for contact resistance variability)
`-i <list_of_r>`	List containing the reference current values. Activated when sIFM is set to `"crv"`. (List of real numbers, only needed for contact resistance variability)
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file "mos_ac_des.plt" -name Data(AC)
set adgs [get_variable_data a(ndrain,ngate) -dataset Data(AC)]

ifm::GetsIFMStdDev -out sigIV -sIFM "mos_rdf_I_ndrain.csv" -y $adgs

create_plot -1d -name Sig
set_plot_prop -plot Sig -title "Standard Deviations"

create_variable -name sigId(stat) -dataset Data(AC) -values $sigIV(I)
create_curve -name sigId(stat) -dataset Data(AC) \
   -axisX "v(ngate)" -axisY "sigId(stat)"
create_variable -name sigVg(stat) -dataset Data(AC) -values $sigIV(V)
create_curve -name sigVg(stat) -dataset Data(AC) \
   -axisX "v(ngate)" -axisY2 "sigVg(stat)"
```

ifm::GetSNM

Computes the static noise margins (SNMs) from butterfly curves for one or more randomization sources.

This procedure takes as input the voltage transfer characteristics (VTC) curves of the left and the right inverters of an SRAM cell. One plot of all VTC curves is known as a *butterfly curve*. The left (right) SNM is defined as the axis-aligned biggest square that can be fitted into the left (right) lob of the butterfly curve. The effective SNM is defined as the smaller value of the two.

Syntax

```
ifm::GetSNM -out <array_name> -squares <array_name> -vtc_left <array_name>
   -vtc_right <array_name> -ncol <var_name> [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of a string-indexed array to store the results. The elements are `left`, `right`, and `eff` for the left, right, and effective SNMs. Each array entry contains a list of the respective SNM values for all randomizations. (Array name, no default)
`-squares <array_name>`	Name of a three-indexed array to store the fitted squares representing the SNM in the butterfly curve. The first index is integer valued. The elements are `1` or `2` for the square of the left or right lob of the butterfly curve. The second index is string valued. The elements are `X` or `Y` for the x-values and y-values of the fitted square. The third index is integer valued and represents the randomization index. (Array name, no default)
`-vtc_left <array_name,` `-vtc_right <array_name>`	Names of two-indexed arrays containing the left and right VTC curves. The first index is string valued. The elements are `Header` or `Data`, where the `Header` contains the names of the columns, such as `Vi(0)`. The corresponding `Data` field contains a list of voltage values. The second index is integer valued and represents the randomization index. (Array name, no default)
`-ncol <var_name>`	Name of a variable containing the number of randomizations in the VTC data. (Variable name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
ifm::ReadCSV -out VTC_L -file Left_VTC.csv  -ncol Ncol
ifm::ReadCSV -out VTC_R -file Right_VTC.csv -ncol Ncol
set j 42
set i_in [expr 2*$j]
set i_ot [expr 2*$j+1]

create_variable -name Vi($j) -dataset VTC(L) -values $VTC_L(Data,$i_in)
create_variable -name Vo($j) -dataset VTC(L) -values $VTC_L(Data,$i_ot)
create_curve -name VTC(L,$j) -dataset VTC(L) -axisX "Vi($j)" -axisY "Vo($j)"

create_variable -name Vi($j) -dataset VTC(R) -values $VTC_R(Data,$i_ot)
create_variable -name Vo($j) -dataset VTC(R) -values $VTC_R(Data,$i_in)
create_curve -name VTC(R,$j) -dataset VTC(R) -axisX "Vi($j)" -axisY "Vo($j)"

ifm::GetSNM -out SNM -squares SQ -vtc_left VTC_L -vtc_right VTC_R -ncol Ncol

create_variable -name SQ1_x($j) -dataset SQs -values $SQ(1,X,$j)
create_variable -name SQ1_y($j) -dataset SQs -values $SQ(1,Y,$j)
create_variable -name SQ2_x($j) -dataset SQs -values $SQ(2,X,$j)
create_variable -name SQ2_y($j) -dataset SQs -values $SQ(2,Y,$j)

create_curve -name SQ1($j) -dataset SQs -axisX "SQ1_x($j)" -axisY "SQ1_y($j)"
create_curve -name SQ2($j) -dataset SQs -axisX "SQ2_x($j)" -axisY "SQ2_y($j)"

puts "Left      SNM: [lindex $SNM(left)  $j]"
puts "Right     SNM: [lindex $SNM(right) $j]"
puts "Effective SNM: [lindex $SNM(eff)   $j]"
```

ifm::GetSRAMVTC

Constructs randomized VTC curves for an SRAM cell for one or more randomization sources.

To compute the randomized VTC of an inverter from an SRAM cell, a method similar to the weighted method outlined in ifm::GetMOSIVs on page 402 for the single transistor is used. Unlike in a single transistor, in an SRAM cell, the output node is not connected to an external voltage source and, therefore, Kirchhoff's law requires that $dI_{\nu,o} = 0$.

For the extraction of SRAM SNMs, you cannot require that the output small-signal voltage variation vanishes because, during the *read* operation of the SRAM cell (access transistor is switched on) in the region of low output bias (PMOS transistor is switched off), the actual voltage of the output node is defined by the voltage divider formed by the access transistor and the NMOS transistor. The current flow fluctuations through the access transistor cannot be adequately compensated by adjusting the gate voltage of the NMOS (and PMOS) transistor. A solution for the ν-th randomized SRAM cell for which all currents through the inverters are the same, as in the reference SRAM cell, while additionally requiring that the voltage at the output is also the same in the reference inverter would not be physical. For example, for certain bias conditions, unrealistically large gate voltage adjustments would be needed to overcompensate the random dopant fluctuation effects in the access transistor.

To obtain physically relevant VTC curves, both the output and input voltages are adjusted:

$$\delta I_{\nu,o} = -y_{o,i} dV_{\nu,i} - y_{o,o} dV_{\nu,o} \tag{33}$$

A method, which results in physical VTC curves, consists of adjusting both $dV_{\nu,i}$ and $dV_{\nu,o}$ based on an automatic analysis of the current flow in the reference device. For this method, the voltage variations $dV_{\nu,i}$ and $dV_{\nu,o}$ are expressed in terms of voltage variations in the coordinate system that is rotated:

$$\begin{bmatrix} dV_{\nu,o} \\ dV_{\nu,i} \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} dV_{\nu,1} \\ dV_{\nu,2} \end{bmatrix} \tag{34}$$

In the rotated coordinate system, the boundary condition is imposed such that $dV_{\nu,1} = 0$, resulting in:

$$dV_{\upsilon,2} = -\frac{\delta I_{\nu,o}}{y_{o,i}\sin(\varphi) + y_{o,o}\cos(\varphi)} \tag{35}$$

At each bias point, an angle is selected that ensures a monotonous and physical solution.

For example, you can find out whether you are in the *hold*-like inverter regime (access transistor is closed) or in the voltage divider regime (PMOS transistor is closed) by monitoring

the current flows in the reference SRAM cell. The reference current through the inverter is the sum of the currents through the PMOS and the access transistors. If the main contribution comes from the PMOS device, you are in the hold-like inverter regime, and you set φ to $\pi/2$. On the other hand, if the main contribution comes from the access transistor, you are in the voltage divider regime, and you set φ to 0. Therefore, you can use the reference current to select the appropriate angle and then compute $dV_{\upsilon,2}$.

An example of a script to compute the current-controlled angle is:

```
set Ips [get_variable_data "$pSOURCE TotalCurrent" -dataset Data(DC)]
set Ias [get_variable_data "$aDRAIN  TotalCurrent" -dataset Data(DC)]
foreach Ip $Ips Ia $Ias {
   set It [expr $Ip + $Ia]
   lappend fis [expr 0.5*$pi*$Ip/$It]
}
```

Here, `pSOURCE` points to the source contact of the PMOS transistor, and `aDRAIN` points to the drain contact of the access transistor of the inverters of interest in the SRAM cell.

Syntax

```
ifm::GetSRAMVTC -out <array_name> -vin <list_of_r> -vout <list_of_r>
   -fi <list_of_r> -aoi <list_of_r> -aoo <list_of_r> -id <string>
   -sifm <array_name> -nrow <var_name> -ncol <var_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of a two-indexed array to store the constructed VTC curves. The first index is string valued. The elements are `Header` or `Data`, where the `Header` contains the names of the columns, such as `Vi(0)`. The corresponding `Data` field contains a list of voltage values. The second index is integer valued. It represents the randomization index. (Array name, no default)
`-vin <list_of_r>`	List containing the reference VTC input voltage values. (List of real numbers, no default)
`-vout <list_of_r>`	List containing the reference VTC output voltage values. (List of real numbers, no default)
`-fi <list_of_r>`	List containing the current-controlled angle values. (List of real numbers, no default)
`-aoi <list_of_r>`	List containing the reference ReY(out,in) matrix values. (List of real numbers, no default)
`-aoo <list_of_r>`	List containing the reference ReY(out,out) matrix values. (List of real numbers, no default)
`-id <string>`	ID of the sIFM variability source. (String, no default)

`-sifm <array_name>`	Name of an array that contains the sIFM data. The array has three indices: The first index is string valued. The elements are the variability source identifiers. The second index is integer valued and represents the row or bias index. The third index is integer valued and represents the column or randomization index. The array element values contain the sIFM linear current response. (Array name, no default)
`-nrow <var_name>`	Name of a variable containing the number of rows (bias points) in the sIFM data. (Variable name, no default)
`-ncol <var_name>`	Name of a variable containing the number of columns (randomizations) in the sIFM data. (Variable name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
load_file sys_des.plt     -name SYSTEM
load_file SRAM_des.plt    -name DC
load_file sram_ac_des.plt -name AC

set Vins [get_variable_data v($IN)       -dataset SYSTEM]
set Vots [get_variable_data v($OUT)      -dataset SYSTEM]
set aois [get_variable_data a($OUT,$IN)  -dataset AC]
set aoos [get_variable_data a($OUT,$OUT) -dataset AC]
set Ips  [get_variable_data "SourceP2 TotalCurrent"  -dataset DC]
set Ias  [get_variable_data "DrainACC2 TotalCurrent" -dataset DC]

foreach Ip $Ips Ia $Ias {
   set It [expr $Ip + $Ia]
   lappend fis [expr 0.5*$ifm::pi*$Ip/$It]
}

ifm::ReadsIFM -out sIFM -nrow Nrow -ncol Ncol \
   -files "flipL_n12_sram_rdf_I_OR.csv" -ids "rdf"

ifm::GetSRAMVTC -out VTC -id "rdf" -vin $Vins -vout $Vots \
   -fi $fis -aoi $aois -aoo $aoos -sifm sIFM -nrow Nrow -ncol Ncol

create_plot -1d -name RanVTC
select_plots RanVTC
set j 42
set i_in [expr 2*$j]
set i_ot [expr 2*$j+1]
```

```
create_variable -name Vi($j) -dataset RanVTC(rdf) -values $VTC(Data,$i_in)
create_variable -name Vo($j) -dataset RanVTC(rdf) -values $VTC(Data,$i_ot)

create_curve -name VTC($j) -dataset RanVTC(rdf) \
   -axisX "Vi($j)" -axisY "Vo($j)"
```

ifm::ReadCSV

Reads a CSV file, and passes the read data to the calling program in the form of a Tcl array.

NOTE The CSV files are assumed to have the following format: One header line containing the names of the datasets (no whitespace) followed by a number of rows containing the values in the dataset, for example, `-1.11e-12,3.92e-14,-1.66e-13,-6.09e-13,...` (no whitespace).

Syntax

```
ifm::ReadCSV -out <array_name> -file <string> -ncol <var_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array with two indices to store the read data. The first index is string valued and contains the elements `Header` and `Data`. The second index is integer valued and enumerates the number of columns in the CSV file. The values of the `Header` elements are the dataset names. The values of the `Data` elements contain lists with the values in the dataset. (Array name, no default)
`-file <string>`	Name of the CSV file to be read. (String, no default)
`-ncol <var_name>`	Name of a variable containing the number of columns found in the CSV file. (Variable name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Ncol 3
set CSV(Header,0) "A"
set CSV(Header,1) "B"
set CSV(Header,2) "C"
set CSV(Data,0) [list 1.1 1.2 1.3 1.4]
set CSV(Data,1) [list 2.1 2.2 2.3 2.4]
set CSV(Data,2) [list 3.1 3.2 3.3 3.4]
ifm::WriteCSV -csv CSV     -ncol Ncol     -file "my.csv"
ifm::ReadCSV  -out ReadCSV -ncol ReadNcol -file "my.csv"
for {set icol 0} {$icol < $ReadNcol} {incr icol} {
   puts "Column name is: $ReadCSV(Header,$icol)"
   puts "Column data is: $ReadCSV(Data,$icol)"
}
Column name is: A
Column data is: 1.1 1.2 1.3 1.4
Column name is: B
Column data is: 2.1 2.2 2.3 2.4
Column name is: C
Column data is: 3.1 3.2 3.3 3.4
```

ifm::ReadsIFM

Reads one or more sIFM CSV files containing the linear current responses, and passes the read data to the calling program in the form of a Tcl array.

This procedure also supports the computation of the linear current responses for contact resistance variability.

NOTE The CSV files are assumed to have the following format: One header line containing the names of the datasets (no whitespace) followed by a number of rows containing the values in the dataset, for example, `-1.11e-12,3.92e-14,-1.66e-13,-6.09e-13,...` (no whitespace).

Syntax

```
ifm::ReadsIFM -out <array_name> -files <list_of_strings>
   -ids <list_of_strings> -nrow <var_name> -ncol <var_name>
   [-rsig <r> -nrand <i> -rseed <i> -ydd <list_of_r>] -i <list_of_r>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the read data. The array has three indices: The first index is string valued. The elements are the variability source identifiers. The second index is integer valued and represents the row or bias index. The third index is integer valued and represents the column or randomization index. The array element values contain the sIFM linear current response. (Array name, no default)
`-files <list_of_strings>`	List containing the names of the sIFM CSV data files. (List of strings, no default)
`-ids <list_of_strings>`	List containing the variability source identifiers. If the list contains the special identifier `SUM`, the combined data from all variability sources will also be computed. (List of strings, no default)
`-nrow <var_name>`	Name of a variable to store the number of rows (bias points) found in the sIFM CSV file. (Variable name, no default)
`-ncol <var_name>`	Name of a variable to store the number of columns (randomizations) found in the sIFM CSV file. (Variable name, no default)
`-rsig <r>`	Standard deviation of contact resistance variability in Ohm. Activated when ids contains `"crv"`. (Real, only needed for contact resistance variability)
`-nrand <i>`	Number of random samples. Activated when ids contains `"crv"`. (Integer, only needed for contact resistance variability)

`-rseed <i>`	Random number seed. Activated when ids contains `"crv"`. (Integer between 1 and 2147483647; only needed for contact resistance variability)
`-ydd <list_of_r>`	List containing the reference Y(d,d) matrix elements. Activated when ids contains `"crv"`. (List of real numbers, only needed for contact resistance variability)
`-i <list_of_r>`	List containing the reference current values. Activated when ids contains `"crv"`. (List of real numbers, only needed for contact resistance variability)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set IDs   [list rdf ift SUM]
set FILEs [list rdf_I_ndrain.csv ift_I_ndrain.csv SUM_I_ndrain.csv]

ifm::ReadsIFM -out sIFM -nrow Nrow -ncol Ncol -files $FILEs -ids $IDs

puts "The linear drain current response due to random dopant fluctuations in
the 42th randomization at 12th bias point is: sIFM(rdf,12,42) =
$sIFM(rdf,12,42).  The corresponding response to interface traps is
sIFM(ift,12,42) = $sIFM(ift,12,42). The combined response is sIFM(SUM,12,42) =
$sIFM(SUM,12,42)"
```

NOTE The CSV file associated with the `SUM` ID is not actually read and, therefore, it does not have to exist. The actual dataset is computed automatically by summing all previously read datasets.

ifm::WriteCSV

Writes a Tcl array to a CSV file.

Syntax

```
ifm::WriteCSV -file <string> -csv <array_name> -ncol <var_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-file <string>`	Name of a CSV file to be written. (String, no default)
`-csv <array_name>`	Name of an array with two indices, containing the data to be written. The first index is string valued and contains the elements `Header` and `Data`. The second index is integer valued and enumerates the number of columns in the CSV file. The values of the `Header` elements are the dataset names. The values of the `Data` elements contain lists with the values in the dataset. (Array name, no default)
`-ncol <var_name>`	Name of a variable containing the number of columns in the CSV data to be written. (Variable name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Ncol 3
set CSV(Header,0) "A"
set CSV(Header,1) "B"
set CSV(Header,2) "C"
set CSV(Data,0) [list 1.1 1.2 1.3 1.4]
set CSV(Data,1) [list 2.1 2.2 2.3 2.4]
set CSV(Data,2) [list 3.1 3.2 3.3 3.4]

ifm::WriteCSV -csv CSV -ncol Ncol -file "my.csv"
```

lib::SetInfoDef

Sets the default information level.

NOTE Level 0: Warning, error, or status messages only.
Level 1: Echo results.
Level 2: Show progress and some debug information.
Level 3: Show all debug information.

The local info level also can be set using the `-info` keyword of the procedures in the IFM library.

Syntax

```
lib::SetInfoDef 0 | 1 | 2 | 3
```

Argument	Description
`<info_level>`	Sets the default info level. Default: `0`

Returns

None.

Example

```
lib::SetInfoDef 2
```

APPENDIX G Fitting Dispersive Model Parameters for EMW

This appendix describes a set of procedures to fit dispersive model parameters for Sentaurus Device Electromagnetic Wave Solver (EMW).

Overview

The `emw::fit` library enables users to fit wavelength-dependent complex refractive index (CRI) data to obtain a corresponding dispersive model parameter file for subsequent EMW simulations.

The `emw::fit` library checks the availability of an EMW license, but it does not check out any EMW license.

The library utilizes a simplex search method [1] as the core algorithm to find the minimum of the residue of the CRI.

The `emw::fit` library is loaded automatically when Sentaurus Visual starts. However, if you have disabled the automatic loading of extension libraries, you can load the `emw::fit` library explicitly with the command:

```
load_library emw_fit
```

General Flow

First, you provide a parameter file (`*.par`) containing the material CRI table to be fitted. The format follows the Sentaurus Device style:

```
Material = "Silicon" {
   ComplexRefractiveIndex {
      Formula = 1
      NumericalTable (
         0.1908 0.84 2.73;
         ...
      )
   }
}
```

Second, the fitting is set up using the `emw::fit` procedures such as:

- `ComplexRefractiveIndex` to set the CRI models and the material.
- `DispersiveMedia` to specify the dispersive models, the starting values, and the value limits.
- `Fitting` to set the fitting parameters such as maximum iteration, wavelength range, and weight function.
- `Globals` to define the input and output files.
- `Plot` to control at which iteration steps the fit curve is plotted.

Third, the fit is performed with `emw::fit::Run`. At the end of the fit, a result file containing the fit curves and a dispersive model parameter file to be used in subsequent EMW simulations is written to disk, according to the `Globals` settings. Typical file names are `n1_001_fit.plt` for the fit curves and `n1_001_fit.par` for the parameter file.

Upon completion of the fitting, the results must be checked visually using the procedure `emw::fit::Graph`. If the results are acceptable, the output parameter file can be used for a subsequent EMW simulation (see Sentaurus™ Device Electromagnetic Wave Solver User Guide, Specifying User-Defined Dispersive Model Poles in Parameter File on page 39).

If the results are not acceptable, you must modify the fitting setup and rerun the fit. Calling `emw::fit::Run` again will increment the index in the result file and the dispersive model parameter file, such that a subsequent `emw::fit::Graph` procedure will allow you to compare the different fit sessions along with the reference data. In addition, the procedure `emw::fit::Graph` prints out the coefficient of determination R^2 for the imaginary and the real parts of the dispersive function separately:

```
emw::fit::Graph
# R2(Real): 0.999673
# R2(Imag): 0.999626
```

The closer the value of R^2 to 1, the better is the fit.

To check the current fit settings, issue a command without any arguments in the Tcl Command pane of Sentaurus Visual, for example:

```
emw::fit::Globals
# ResultFile = n1_basic_fit.plt
# CRITableFile = par/Aluminum.par
# ParameterFile = n1_basic_fit.par
# LogFile = n1_basic_fit.log
```

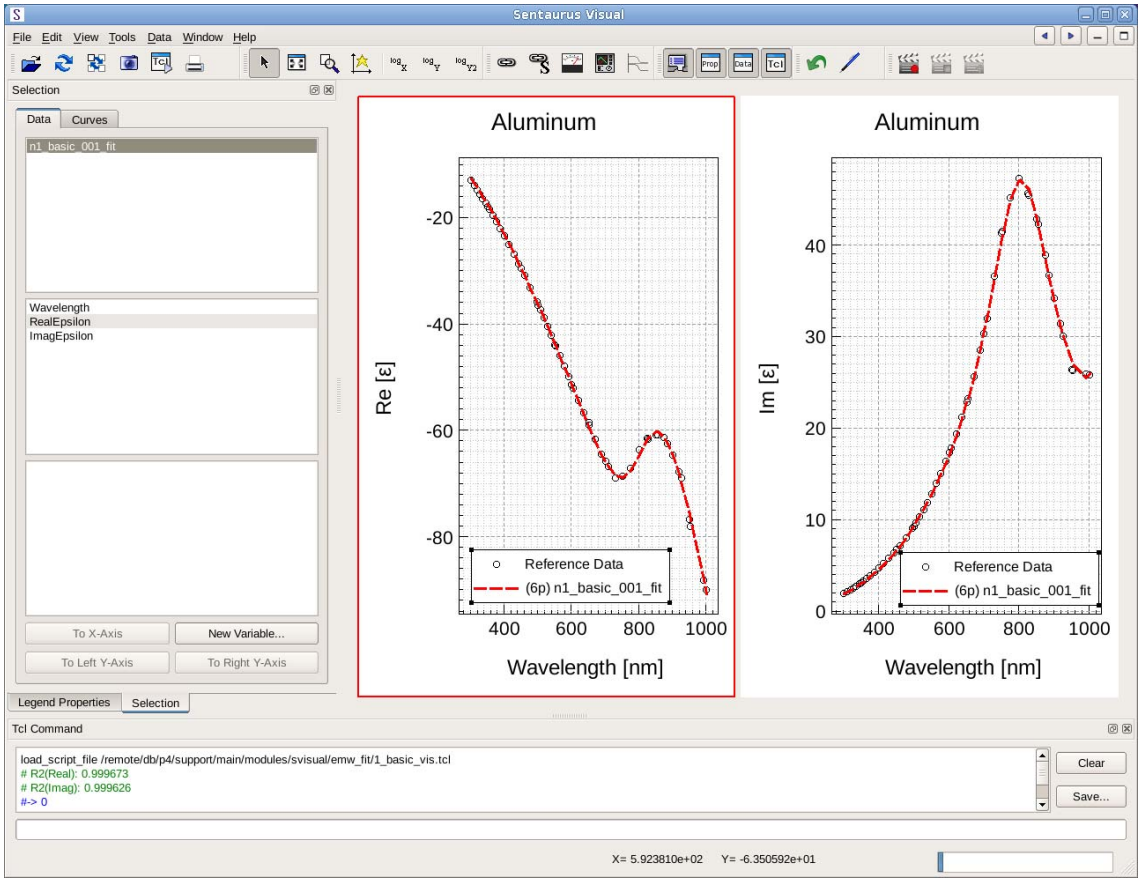


Figure 118 Visual inspection of current fit settings

NOTE It is recommended that users run a small test of EMW after obtaining the fitted parameters to ensure stability before running EMW using a large simulation domain.

NOTE To reduce the interactive typing effort, you can issue the following command that will allow you to call the procedures without the namespace specification, for example, `Globals` instead of `emw::fit::Globals`:

```
namespace import emw::fit::*
```

An example project with more detailed step-by-step instructions is provided in the TCAD Sentaurus Tutorial, Sentaurus Device Electromagnetic Wave Solver module.

emw::fit::Clear

Removes all curves from the plot.

Syntax

```
emw::fit::Clear [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

emw::fit::ComplexRefractiveIndex

Sets the parameters related to the complex refractive index (CRI) model. You must specify a material. If called without arguments, this procedure prints the currently set parameters.

Syntax

```
emw::fit::ComplexRefractiveIndex
   -Material "<string>"
   [-WavelengthDep <identifier> | {<identifier> ...}]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-Material "<string>"`	Name of material to which dispersive media is fitted.
`-WavelengthDep <identifier> | {<identifier> ...}`	Controls the wavelength dependency of the real and imaginary parts of the complex refractive index. Options: `Real`, `Imag`. Default: `{Real Imag}`
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

emw::fit::DispersiveMedia

Specifies the type and number of poles for the dispersive media used for the fitting. It also specifies dispersive model–specific parameters in the triplet format of (start_value, min_value, max_value). See Sentaurus™ Device Electromagnetic Wave Solver User Guide, Specifying User-Defined Dispersive Model Poles in Parameter File on page 39 for dispersive model–specific parameters. If called without arguments, this procedure prints the currently set parameters.

Syntax

```
emw::fit::DispersiveMedia
   [-DebyeAmp {<float> <float> <float>}]
   [-DebyeRelaxTime {<float> <float> <float>}]
   [-DrudeDampFac {<float> <float> <float>}]
   [-DrudeFreq {<float> <float> <float>}]
   [-EpsilonInf {<float> <float> <float>}]
   [-LorentzAmp {<float> <float> <float>}]
   [-LorentzDampFac {<float> <float> <float>}]
   [-LorentzFreq {<float> <float> <float>}]
   [-Model <identifier>]
   [-ModLorentzAmp {<float> <float> <float>}]
   [-ModLorentzDampFac {<float> <float> <float>}]
   [-ModLorentzDampFac2 {<float> <float> <float>}]
   [-ModLorentzFreq {<float> <float> <float>}]
   [-NumberDebyePoles <integer>]
   [-NumberDrudePoles <integer>]
   [-NumberLorentzPoles <integer>]
   [-NumberModLorentzPoles <integer>]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-DebyeAmp {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the Debye amplitude.
`-DebyeRelaxTime {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the Debye relaxation time. Unit: s.
`-DrudeDampFac {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the Drude damping factor. Unit: $\mathrm{rad\,s^{-1}}$.
`-DrudeFreq {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the Drude frequency. Unit: $\mathrm{s^{-1}}$.
`-EpsilonInf {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of ε at frequency infinity.

`-LorentzAmp {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the Lorentz amplitude.
`-LorentzDampFac {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the Lorentz damping factor. Unit: $\mathrm{rad\,s^{-1}}$.
`-LorentzFreq {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the Lorentz frequency. Unit: $\mathrm{s^{-1}}$.
`-Model <identifier>`	Selects the dispersive model. Options: • `Debye` • `Drude` • `DrudeLorentz` • `DrudeModLorentz` • `Lorentz` • `ModLorentz`
`-ModLorentzAmp {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the ModLorentz amplitude.
`-ModLorentzDampFac {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the ModLorentz damping factor. Unit: $\mathrm{rad\,s^{-1}}$.
`-ModLorentzDampFac2 {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the second ModLorentz damping factor. Unit: $\mathrm{rad\,s^{-1}}$.
`-ModLorentzFreq {<float> <float> <float>}`	Specifies the (start, minimum, maximum) values of the ModLorentz frequency. Unit: $\mathrm{s^{-1}}$.
`-NumberDebyePoles <integer>`	Specifies the number of Debye poles. Default: `1`
`-NumberDrudePoles <integer>`	Specifies the number of Drude poles. Default: `1`
`-NumberLorentzPoles <integer>`	Specifies the number of Lorentz poles. Default: `1`
`-NumberModLorentzPoles <integer>`	Specifies the number of ModLorentz poles. Default: `1`
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

emw::fit::Fitting

Sets fitting parameters. If called without arguments, this procedure prints the currently set parameters.

Syntax

```
emw::fit::Fitting
   [-CRITableStep <integer>] [-MaxIterations <integer>]
   [-SimplexMaxFunEvals <integer>] [-SimplexMaxIterations <integer>]
   [-SimplexTolFun <float>] [-SimplexTolX <float>]
   [-WavelengthRange {<float> <float>}] [-WeightImagEps <float>]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-CRITableStep <integer>`	Controls fitting every `<integer>` data entries in the complex refractive index table. Default: `1`
`-MaxIterations <integer>`	Maximum number of fitting iterations. Default: `100`
`-SimplexMaxFunEvals <integer>`	Maximum number of objective function evaluations within a simplex search. Default: `5000`
`-SimplexMaxIterations <integer>`	Maximum number of iterations within a simplex search. Default: `5000`
`-SimplexTolFun <float>`	Convergence tolerance of objective function within a simplex search. Default: `0.0001`
`-SimplexTolX <float>`	Convergence tolerance of parameters within a simplex search. Default: `0.0001`
`-WavelengthRange {<float> <float>}`	Specifies the wavelength range to be used for the fitting given by an interval (min_wavelength, max_wavelength). If no interval is given, the entire wavelength range of the complex refractive index table is used. Unit: nm.
`-WeightImagEps <float>`	Weight on the fitting error of imaginary epsilon. Default: `2`
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

emw::fit::Globals

Sets all the global parameter settings for the simulation, such as input and output file names. If called without arguments, this procedure prints the currently set parameters.

Syntax

```
emw::fit::Globals
   [-CRITableFile "<string>"] [-LogFile "<string>"]
   [-ParameterFile "<string>"] [-ResultFile "<string>"]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-CRITableFile "<string>"`	Name of the input parameter file containing the complex refractive index table. Default: `cri_table.par`
`-LogFile "<string>"`	Name of the log file. By default, the name of the command file is used with the extension `.log`. If the name ends with `.Z` or `.gz`, a compressed log file will be written.
`-ParameterFile "<string>"`	Name of the output parameter file for the fitting results. Default: `fitting_results.par`
`-ResultFile "<string>"`	Name of the file for visualizing the fitting results. Default: `fitting_results.plt`
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

emw::fit::Graph

Plots the results of a previous fit performed with the `emw::fit::Run` procedure.

Syntax

```
emw::fit::Graph [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

emw::fit::Plot

Specifies at which iterations the fitted complex refractive index (CRI) data is written in the result file. This is useful when users need to investigate the sequence of convergence. In most cases, the last fitting iteration offers the best fitting result in terms of minimizing the residue of the CRI. If called without arguments, this procedure prints the currently set parameters.

Syntax

```
emw::fit::Plot
   [-EndTick <integer>] [-FinalPlot <identifier>]
   [-StartTick <integer>] [-TickStep <integer>]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-EndTick <integer>`	Specifies the end of the active tick step interval. Default: ∞
`-FinalPlot <identifier>`	Controls whether a final plot is created at the end of the fitting. Options: `Yes`, `No`. Default: `Yes`
`-StartTick <integer>`	Specifies the beginning of the active tick step interval. Default: ∞
`-TickStep <integer>`	Plotting is performed every `<integer>` tick steps only. Default: `10`
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

emw::fit::Run

Executes a fit. To set up the fit, use the following procedures: `ComplexRefractiveIndex`, `DispersiveMedia`, `Fitting`, `Globals`, and `Plot`.

Syntax

```
emw::fit::Run [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

References

[1] J. C. Lagarias *et al.*, “Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions,” *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 112–147, 1998.

APPENDIX H Two-Port Network RF Extraction Library

This appendix provides information about the procedures of the RF extraction library.

Under the assumption that a transistor can be modeled by a two-port network, the procedures of the two-port network radio frequency (RF) extraction library are used to compute:

- RF parameters from AC analysis data.
- Noise parameters from noise analysis data.

The functionality of the RF extraction library includes:

- RF matrix conversion: Converting an admittance (Y-)matrix to a hybrid (h-)matrix, a scattering (S-)matrix, and an impedance (Z-)matrix.
- Plotting the small-signal data (conductance, capacitance, and RF parameters). In addition to rectangular plots, polar plots and Smith charts are supported.
- Plotting the following noise spectral densities (NSDs) or power spectral densities (PSDs) of various representations (noise equivalent circuits) of a noisy transistor:
  - Noise voltage spectral density (NVSD) for impedance representation.
  - Noise current spectral density (NISD) for admittance representation.
- RF parameter extraction:
  - Computing small-signal current gain, stability criteria such as the Rollett stability factor and the stability condition delta, various power gains such as maximum available gain (MAG), maximum stable gain (MSG), Mason's unilateral gain (MUG), and unilateral figure of merit.
  - Extracting transistor figures of merit such as cutoff frequency, maximum frequency of oscillation, and cutoff frequency for stability.
- Noise parameter extraction:
  - Computing PSDs of the equivalent input noise generators in chain representation of a noisy transistor.
  - Computing the noise figure of a transistor.
  - Extracting various noise parameters of a transistor such as the minimum noise figure, the equivalent noise resistance and conductance, and the optimum source admittance and impedance.

- Complex arithmetic support (both scalar and vectorial versions).
- Exporting data in comma-separated value (CSV) file format.

The RF extraction library is loaded with the command:

```
load_library rfx
```

Syntax Conventions

The RF extraction library uses a unique namespace identifier (`rfx::`) for its procedures. All procedures and variables associated with this library are called with the namespace identifier prepended, for example:

```
rfx::<proc_name>
```

Each procedure has several arguments. The RF extraction library uses an input parser that accepts arguments of the form:

```
-keyword <value>
```

NOTE All Sentaurus Visual libraries support the standard Sentaurus Visual syntax in which keywords are preceded by a dash. For backward compatibility, all Sentaurus Visual libraries continue to support the `keyword= <value>` syntax as well. For each procedure call, you can use either the `-keyword <value>` syntax or the `keyword= <value>` syntax. However, within any one procedure call, only one type of syntax can be used. Only the new syntax is documented. If you want to continue using the `keyword= <value>` syntax, you also can insert whitespace between the keyword and the equal sign, for example, `keyword = <value>`. Omitting the whitespace between the equal sign and the value field will result in a failure if the value is a de-referenced Tcl variable. Use `keyword= $val` (*not* `keyword=$val`).

The parser accepts arguments in any order. For some arguments, default values are predefined. Such arguments may be omitted. If arguments for which no defaults are predefined are omitted, the procedure will exit with an error message. In addition, unrecognized arguments result in an error message.

Instead of using the standard Tcl method of using the return value of the procedure to pass results back to the calling program, the RF extraction library uses a *passing-by-reference* method to return the results to the calling program. The procedure keyword `-out` is used to pass the results back to the calling program:

```
-out <var_name>, <list_name>, or <array_name>
```

The following conventions are used for the syntax of Tcl commands:

- Angle brackets – <> – indicate text that must be replaced, but they are *not* part of the syntax. In particular, the following type identifiers are used:
  - `<r>`: Replace with a real number, or a de-referenced Tcl variable that evaluates to a real number. For example: `$val`.
  - `<i>`: Replace with an integer, or a de-referenced Tcl variable that evaluates to an integer. For example: `$i`.
  - `<string>`: Replace with a string, or a de-referenced Tcl variable that evaluates to a string. For example: `$file`.
  - `<list_of_r>`: Replace with a list of real numbers, or a de-referenced Tcl variable that evaluates to a list of real numbers. For example: `$values`.
  - `<list_of_strings>`: Replace with a list of strings, or a de-referenced Tcl variable that evaluates to a list of strings. For example: `$files`.
  - `<var_name>`: Replace with the *name* of a local Tcl variable.
    For example: `val` (*not* `$val`).
  - `<list_name>`: Replace with the *name* of a local Tcl list.
    For example: `values` (*not* `$values`).
  - `<array_name>`: Replace with the *name* of a local Tcl array.
    For example: `myarray` (*not* `$myarray`).
  - `<dataName>`: Replace with the *name* of a dataset.
  - `<fileName>`: Replace with the *name* of a file, or a de-referenced Tcl variable that evaluates to the name of a file.
  - `<plotName>`: Replace with the *name* of a plot.
- Brackets – [] – indicate that the argument is optional, but they are *not* part of the syntax.
- A vertical bar – | – indicates options, only one of which can be specified.

Help for Procedures

To request help on a specific procedure, set the `-help` keyword to `1`:

```
rfx::<proc_name> -help 1
```

If this command is included in a Sentaurus Visual file, when Sentaurus Visual is executed in:

- Batch mode in Sentaurus Workbench, the help information is printed to the runtime output file (with the extension `.out`) of the corresponding Sentaurus Visual node.

- Interactive mode in Sentaurus Workbench, the help information is displayed in the Tcl Command panel as well as printed in the Sentaurus Visual runtime output file.

You also can type this command in the Tcl Command panel of the graphical user interface, in which case, the help information is displayed in the same panel.

Output of Procedures

As discussed in Syntax Conventions on page 438, all procedures of the RF extraction library pass the results back to the calling program by storing the results in a Tcl variable. The name of this Tcl variable is specified as the value of the `-out` keyword.

If there are errors in the RF extraction library procedures, the behavior of Sentaurus Visual depends on whether it is executed in batch mode or interactive mode in Sentaurus Workbench. In batch mode, Sentaurus Visual exits and an error message is printed only in the Sentaurus Visual error file (with the extension `.err`). In interactive mode, the error message is displayed in the Tcl Command panel as well as printed in the Sentaurus Visual runtime error file.

All procedures also print several messages (including warning messages). If Sentaurus Visual is executed in batch mode, the messages are printed only in the Sentaurus Visual output file; whereas, in interactive mode, the messages are displayed in the Tcl Command panel as well as printed in the Sentaurus Visual runtime output file.

The amount of information printed depends on the information level specified by the procedure `lib::SetInfoDef`.

Overview of RF Extraction Library Procedures

For the simulation of RF characteristics and the extraction of RF parameters, small-signal (AC) analysis is performed in Sentaurus Device by varying the bias at a contact and performing a frequency sweep at each bias point. The Sentaurus Device AC data file contains the conductance values a_{ij} and capacitance values c_{ij} at each bias and frequency point for all contact-to-contact combinations included in the small-signal analysis (see A-Matrix, C-Matrix, and Y-Matrix on page 442). The RF extraction library assumes that the transistor can be modeled by a two-port network as shown in Figure 119 on page 443.

The functionality of the RF extraction library and the corresponding procedures are:

- Loading the Sentaurus Device AC data file, and creating a Y-matrix and PSD matrices:
  The Sentaurus Device AC data file is loaded in Sentaurus Visual using the `rfx::Load` procedure, which creates the Tcl array `rfx::AC` containing the conductance and capacitance values (see rfx::Load on page 483). This data also is converted to admittance

or Y-parameters, and the Y-parameters or the Y-matrix are stored in the Tcl array `rfx::Y`. The `rfx::Load` procedure also creates the PSD Tcl arrays (corresponding to PSD matrices) and other variables that are summarized in Table 31 on page 460. For details about these arrays, see A-Matrix, C-Matrix, and Y-Matrix on page 442 and Power Spectral Density Matrices on page 444.

- Converting a Y-matrix to other matrices:
  The Y-matrix is converted to either an h-matrix, an S-matrix, or a Z-matrix (see Matrix Conversions on page 449) using the matrix conversion procedures `rfx::Y2H` (see rfx::Y2H on page 493), `rfx::Y2S` (see rfx::Y2S on page 494), and `rfx::Y2Z` (see rfx::Y2Z on page 495), respectively. All these matrices are complex and the matrix conversion procedures internally use the complex arithmetic procedures (see Complex Arithmetic Support on page 496). The RF parameters Y_{ij}, h_{ij}, S_{ij}, and Z_{ij} are the elements of the Y-, h-, S-, and Z-matrix, respectively.
- Creating Sentaurus Visual datasets containing small-signal data and noise analysis data:
  Sentaurus Visual datasets containing the small-signal data (a_{ij}, c_{ij}, Y_{ij}, h_{ij}, S_{ij}, or Z_{ij}) as a function of frequency or bias can be created using the procedure `rfx::CreateDataset` (see rfx::CreateDataset on page 462). The datasets corresponding to the RF parameters contain the real and imaginary parts, as well as the absolute value and the phase of the RF parameters. The absolute value of these parameters also can be computed in units of decibel (dB). In addition, Sentaurus Visual datasets containing noise analysis data (S_{V}^{ij} and S_{I}^{ij}) can be created using the `rfx::CreateDataset` procedure.
- Plotting small-signal data and noise analysis data:
  The datasets created using the `rfx::CreateDataset` procedure can be used to visualize the small-signal data and noise analysis data as a function of frequency or bias. The RF parameters can be visualized using the following types of plot:
  - Rectangular plots using Sentaurus Visual commands.
  - Polar plots using the `rfx::PolarBackdrop` procedure (see rfx::PolarBackdrop on page 488).
  - Smith charts using the `rfx::SmithBackdrop` procedure (see rfx::SmithBackdrop on page 492).
- Computing power gains and stability criteria:
  Various power gains and stability criteria are computed using S-parameters (see Gains, Amplifier Stability, and Unilateralization on page 450) using the `rfx::GetPowerGain` procedure (see rfx::GetPowerGain on page 480). This procedure also can be used to create datasets containing the power gains and stability criteria, either as a function of frequency or bias.
- Extracting transistor figures of merit:
  The transistor figures of merit such as cutoff frequency, maximum frequency of oscillation, and cutoff frequency for stability (see Transistor Figures of Merit on page 452) can be extracted using the procedures `rfx::GetFt` (see rfx::GetFt on page 472), `rfx::GetFmax` (see rfx::GetFmax on page 470), and `rfx::GetFK1` (see rfx::GetFK1 on

page 468). These procedures can be used to create the datasets containing the extracted figures of merit as a function of bias. The first two procedures also can be used to create datasets containing the derivative of the gain as a function of frequency.

- Extracting transistor noise parameters:
  Transistor noise parameters such as the minimum noise figure, the equivalent noise resistance and conductance, and the optimum source admittance and impedance, along with other noise parameters and noise figures (see Noise Figure of a Linear Two-Port Network on page 456), can be extracted using the `rfx::GetNoiseFigure` procedure (see rfx::GetNoiseFigure on page 475). This procedure can create datasets containing the extracted noise parameters as a function of frequency or bias.
- Exporting small-signal data:
  The small-signal data can be exported to a CSV file using the `rfx::Export` procedure (see rfx::Export on page 466).

Equations Used in RF Extraction Library

A-Matrix, C-Matrix, and Y-Matrix

The Sentaurus Device AC data file contains the conductance matrix (A-matrix, A) and the capacitance matrix (C-matrix, C) for each bias (v) and frequency (f) point for all contact-to-contact combinations included in the small-signal analysis. The rows and columns of these matrices are given by the nodes included in the small-signal analysis.

For a 3D device, the Sentaurus Device AC data file contains the following for each frequency and bias point:

- a_{ij}, the coefficients or elements of the A-matrix
- c_{ij}, the coefficients or elements of the C-matrix

The A-matrix and C-matrix are converted to an admittance matrix (Y-matrix, Y) using:

$$Y = A + j\omega C = A + jB \tag{36}$$

where:

- j is the imaginary unit.
- $\omega = 2\pi f$ is the angular frequency.
- Matrix B is the susceptance matrix, with coefficients b_{ij}.

The RF extraction library assumes that the transistor can be modeled by a two-port network as shown in Figure 119. Therefore, the RF extraction library reads only a 2×2 matrix, corresponding to a two-port network setup. If other ports are present, they are ignored.

Figure 119 Two-port network schematic

For a two-port network, the complex Y-matrix at a particular frequency f and bias point v is represented by:

$$Y = \begin{bmatrix} Y_{11}(f, \mathrm{v}) & Y_{12}(f, \mathrm{v}) \\ Y_{21}(f, \mathrm{v}) & Y_{22}(f, \mathrm{v}) \end{bmatrix} \tag{37}$$

where the elements of the Y-matrix, Y_{ij}, are the admittance (Y-)parameters. The real and imaginary parts of the complex Y-parameters are given by:

$$\Re(Y_{ij}(f, \mathrm{v})) = a_{ij}(f, \mathrm{v}) \tag{38}$$

$$I(Y_{ij}(f, \mathrm{v})) = b_{ij}(f, \mathrm{v}) = \omega c_{ij}(f, \mathrm{v}) \tag{39}$$

Tcl Arrays rfx::AC and rfx::Y

When the Sentaurus Device AC data file is loaded in Sentaurus Visual using the procedure `rfx::Load`, two Tcl arrays `rfx::AC` and `rfx::Y` are created (see Table 32 on page 463). Both these arrays are two dimensional and have the same form:

```
rfx::AC($ReIm,$P1,$P2,$if,$iv)
rfx::Y($ReIm,$P1,$P2,$if,$iv)
```

where:

- `ReIm`: `0` (real part) or `1` (imaginary part):
  - For `rfx::AC`, `0` corresponds to $a_{ij}(f, \mathrm{v})$ and `1` corresponds to $c_{ij}(f, \mathrm{v})$.
  - For `rfx::Y`, `0` corresponds to $a_{ij}(f, \mathrm{v})$ and `1` corresponds to $b_{ij}(f, \mathrm{v})$ (Eq. 39).
- `P1,P2`: `1` or `2` (port number)
- `if: 0-(rfx::i_freqend)` frequency index
- `iv: 0-(rfx::i_biasend)` bias point index

Therefore, the `rfx::AC` array contains the coefficients $a_{ij}(f, \mathrm{v})$ and $c_{ij}(f, \mathrm{v})$ for all frequency and bias points. The `rfx::Y` array contains the coefficients $a_{ij}(f, \mathrm{v})$ and $b_{ij}(f, \mathrm{v})$. To access the small-signal data or the RF parameter for a given bias or frequency, the appropriate array indices (frequency index and bias point index) must be given.

Power Spectral Density Matrices

The effect of a noisy electronic device on a circuit can be analyzed using either a small-signal model of the device or a two-port network approach. RF extraction library implements noise parameter extraction using the two-port network approach.

A noisy device such as a transistor can be represented by a two-port network with internal noise sources (see Figure 120). For small signals, any noisy two-port network can be replaced by a noise equivalent circuit consisting of a noiseless two-port network and two external equivalent noise sources added to the terminals of the two-port (see Figure 120). The external noise sources are either noise voltage sources or noise current sources, and they produce the same noise voltages at the circuit terminals as the internal noise sources.

Power Spectral Densities

As shown in Figure 120, there are several equivalent representations of a noisy two-port network depending on the type of the external noise sources and their arrangement relative to the noiseless two-port [1].

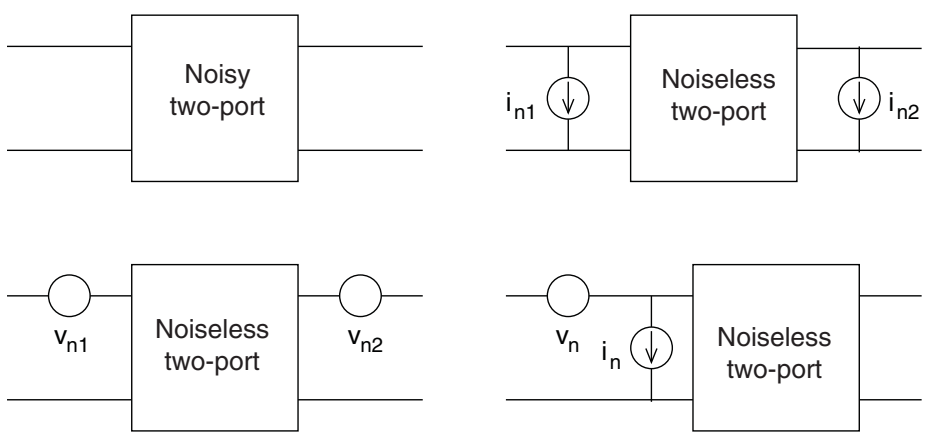


Figure 120 (*Upper left*) Two-port network with internal noise sources, and noise equivalent circuits of a two-port network: (*upper right*) admittance representation, (*lower left*) impedance representation, and (*lower right*) chain representation

The most commonly used representations are:

- Impedance representation: Noise voltage sources v_{n1} and v_{n2} are placed in series with the input and output terminals (see Figure 120 on page 444, lower-left image).
- Admittance representation: Noise current sources i_{n1} and i_{n2} are placed in parallel with the input and output terminals (see Figure 120, upper-right image).
- Chain representation (equivalent input noise representation): Input noise voltage source v_n and input noise current source i_n are placed at the input terminals (see Figure 120, lower-right image).

The noise sources are characterized by a mean square value and a PSD:

- A noise voltage source v_n is characterized by the mean square value $\overline{v_n^2}$ and the NVSD S_{v_n}.
- A noise current source i_n is characterized by the mean square value $\overline{i_n^2}$ and the NISD S_{i_n}.

The NVSD determines the mean square value of the noise voltage source, and the NISD determines the mean square value of the current voltage source within a frequency interval of width 1 Hz [2]. Therefore, the units of NVSD are V^2/Hz or $V^2 s$, and the units of NISD are A^2/Hz or $A^2 s$. The PSD gives the average power P_{av} that the noise source contributes in a 1 Hz bandwidth around frequency f. The PSD spectrum shows how much average power the noise source contributes at each frequency [3].

For a noise voltage source, its average power and mean square voltage over a frequency interval $[f_1, f_2]$ are related to its NVSD by:

$$P_{av} = \overline{v_n^2} = \int_{f_1}^{f_2} S_{v_n} df \tag{40}$$

For a contact pair (i, j) (or for nodes i and j), the NVSD is denoted by S_V^{ij}, and the NISD is denoted by S_I^{ij}. Autocorrelation PSD corresponds to the case when both terminals are the same ($i = j$); whereas, the cross-correlation PSD corresponds to the case when both terminals are different ($i \neq j$).

The following PSDs can be defined for the various representations of the noisy two-port (see PSDs Computed by Sentaurus Device and RF Extraction Library on page 446):

- S_V^{11}: Noise voltage spectral density at the input port in the impedance representation.
- S_V^{22}: Noise voltage spectral density at the output port in the impedance representation.
- S_V^{12} and S_V^{21}: Cross-correlation spectral density between the input and output voltage noise sources (impedance representation).
- S_I^{11}: Noise current spectral density at the input port in the admittance representation.
- S_I^{22}: Noise current spectral density at the output port in the admittance representation.
- S_I^{12} and S_I^{21}: Cross-correlation spectral density between the input and output current noise sources (admittance representation).

- S_{v_n}: Noise voltage spectral density of the input noise voltage source v_n in the chain representation.
- S_{i_n}: Noise current spectral density of the input noise current source i_n in the chain representation.
- $S_{v_n \bar{i}_n}$ and $S_{i_n \bar{v}_n}$: Cross-correlation spectral density of the equivalent input noise voltage and noise current source (chain representation).

The NISDs in the admittance representation are used to compute the noise correlation coefficient between the input and output noise current sources C_{i_n}:

$$C_{i_n} = \frac{S_{\mathrm{I}}^{12}}{\sqrt{S_{\mathrm{I}}^{11} S_{\mathrm{I}}^{22}}} \tag{41}$$

PSDs Computed by Sentaurus Device and RF Extraction Library

As a result of noise analysis on a two-port network containing nodes i ($i = 1$, input port) and j ($j = 2$, output port), Sentaurus Device computes the autocorrelation and cross-correlation spectral densities for both impedance representation (S_{V}^{11}, S_{V}^{21}, and S_{V}^{22}) and admittance representation (S_{I}^{11}, S_{I}^{21}, and S_{I}^{22}), and writes them in the Sentaurus Device AC data file.

The RF extraction library computes the spectral densities S_{V}^{12} and S_{I}^{12}, as well as the spectral densities characterizing the external noise sources in the chain representation (S_{v_n}, S_{i_n}, $S_{v_n \bar{i}_n}$, and $S_{i_n \bar{v}_n}$) (see rfx::GetNoiseFigure on page 475).

The RF extraction library converts the PSD data in the AC data file to PSD Tcl arrays using the `rfx::Load` procedure (see rfx::Load on page 483) and to Sentaurus Visual datasets using the `rfx::CreateDataset` procedure (see rfx::CreateDataset on page 462). Table 34 on page 465 gives examples of PSD matrix coefficients and the corresponding names of PSD variables in the AC data file, the PSD Tcl array name, and the dataset variable name.

The NISDs are computed for the current through the selected circuit nodes, assuming a fixed voltage at these nodes. The NVSDs are computed for the voltages at these nodes, assuming the net current to these nodes is fixed [4].

The NISD is saved as `S_I` and the NVSD is saved as `S_V` in the AC data file. In the case of the autocorrelation coefficient for node i, the NISD is denoted by `S_I(i)`. The cross-correlation coefficients have both real and imaginary parts. The real part of the NISD for nodes i and j is denoted by `ReS_IXI(i,j)`. The imaginary part is denoted by `ImS_IXI(i,j)`. Similar conventions apply to the NVSD.

In addition to the abovementioned NISD and NVSD, Sentaurus Device writes several partial noise spectral densities that describe the contribution of specific noise sources.

For example, `S_V_ee` is the NVSD due to electrons and `S_V_eeDiff` is the electron NVSD due to diffusion noise. For a list of all these spectral densities, refer to the *Sentaurus™ Device User Guide* [4].

Power Spectral Density Tcl Arrays

For the impedance representation of a noisy two-port network (see Power Spectral Densities on page 444), the complex S_V-matrix (NVSD matrix) at a particular frequency f and bias point v is represented by:

$$S_V = \begin{bmatrix} S_V^{11}(f, \mathrm{v}) & S_V^{12}(f, \mathrm{v}) \\ S_V^{21}(f, \mathrm{v}) & S_V^{22}(f, \mathrm{v}) \end{bmatrix} \tag{42}$$

Similarly, the complex S_I-matrix (NISD matrix) at a particular frequency f and bias point v is represented by:

$$S_I = \begin{bmatrix} S_I^{11}(f, \mathrm{v}) & S_I^{12}(f, \mathrm{v}) \\ S_I^{21}(f, \mathrm{v}) & S_I^{22}(f, \mathrm{v}) \end{bmatrix} \tag{43}$$

The coefficients $S_V^{ij}(f, \mathrm{v})$ and $S_I^{ij}(f, \mathrm{v})$ are defined in Power Spectral Densities on page 444.

When the Sentaurus Device AC data file is loaded in Sentaurus Visual using the `rfx::Load` procedure, the PSD Tcl arrays (see Table 32 on page 463) are also created if the file contains PSD data. A Tcl array is created for each PSD data stored in the AC data file. For example, the Tcl arrays `rfx::SV` and `rfx::SI` are created and contain the coefficients $S_V^{ij}(f, v)$ and $S_I^{ij}(f, v)$ (see Table 34 on page 465).

The form of these PSD Tcl arrays is the same as the Tcl arrays `rfx::AC` and `rfx::Y`. For example, the arrays `rfx::SV` and `rfx::SI` have the form:

```
rfx::SV($ReIm,$P1,$P2,$if,$iv)
rfx::SI($ReIm,$P1,$P2,$if,$iv)
```

where:

- For `rfx::SV`, `0` corresponds to $\Re(S_V^{ij}(f, v))$ and `1` corresponds to $\mathrm{I}(S_V^{ij}(f, v))$ (Eq. 42).
- For `rfx::SI`, `0` corresponds to $\Re(S_I^{ij}(f, v))$ and `1` corresponds to $\mathrm{I}(S_I^{ij}(f, v))$ (Eq. 43).

Each PSD array contains the autocorrelation coefficients (both ports are the same, $i = j$) as well as the cross-correlation coefficients (both ports are different, $i \neq j$). Since the autocorrelation values are real, the imaginary part is set to `0`.

The PSD arrays for the local noise source (LNS) are created depending on the specific noise models activated in the Sentaurus Device command file. For example, `rfx::SVeeDiff` is created only if the diffusion LNS is specified in the Sentaurus Device command file.

If there is a named noise specification in the Sentaurus Device command file, this name is prefixed to the name of the PSD Tcl array: `<name>_rfx::SV`. For example, if the name of the noise specification is `diff`, examples of array names are `diff_rfx::SVeeDiff` and `diff_rfx::SV`.

NOTE Only one noise specification is supported per simulation. It can be either named or unnamed.

Device Width Scaling for 2D Structures

A 3D device homogeneous in one of the directions (for example, the z-direction) can be analyzed by using a two-dimensional (2D) device structure. In this case, device simulation can be performed on the 2D device structure and the results for the 3D device can be obtained by multiplying the 2D simulation results (terminal currents, conductance, and capacitance) by the device width in the z-direction, W, and in Eq. 36, you replace:

$$A = WA^{2D} \tag{44}$$

$$B = WB^{2D} \tag{45}$$

where A^{2D} and B^{2D} are the conductance matrix and the susceptance matrix of the 2D device, respectively.

W can be set in one of the following ways:

- In the Sentaurus Device input file using the keyword `AreaFactor` in the `Physics` section.
- During postprocessing using the `rfx::Load` procedure by specifying the keyword `-devicewidth` (see rfx::Load on page 483).

The default value of `AreaFactor` as well as the keyword `-devicewidth` is $1\ \mu m$.

NOTE Avoid applying the scaling twice by using only one of the scaling methods.

Some of the parameters such as h_{21} scale trivially with the device width. For other parameters such as S-parameters or the unilateral figure of merit, the device width is important.

NOTE Not all RF quantities scale linearly with the device width. You must use the keyword `AreaFactor` in the Sentaurus Device command file to take into account the device width scaling for 2D structures.

Matrix Conversions

As discussed in Overview of RF Extraction Library Procedures on page 440, the Y-matrix is converted to either an h-matrix, an S-matrix, or a Z-matrix using the matrix conversion formulas summarized here [5][6].

Converting Y-Matrix to h-Matrix

The complex Y-matrix is converted to the complex h-matrix using the formulas:

$$\begin{aligned} h_{11} &= \frac{1}{Y_{11}} \\ h_{12} &= \frac{-Y_{12}}{Y_{11}} \\ h_{21} &= \frac{Y_{21}}{Y_{11}} \\ h_{22} &= \frac{D_y}{Y_{11}} \end{aligned} \tag{46}$$

with $D_y = Y_{11}Y_{22} - Y_{12}Y_{21}$.

Converting Y-Matrix to S-Matrix

The complex Y-matrix is converted to the complex S-matrix using the formulas:

$$\begin{aligned} S_{11} &= \frac{(1-\bar{Y}_{11})(1+\bar{Y}_{22}) + \bar{Y}_{12}\bar{Y}_{21}}{N_y} \\ S_{12} &= \frac{-2\bar{Y}_{12}}{N_y} \\ S_{21} &= \frac{-2\bar{Y}_{21}}{N_y} \\ S_{22} &= \frac{(1-\bar{Y}_{22})(1+\bar{Y}_{11}) + \bar{Y}_{12}\bar{Y}_{21}}{N_y} \end{aligned} \tag{47}$$

with:

$$\bar{Y}_{ij} = Z_o Y_{ij} \tag{48}$$

where Z_o is the characteristic impedance and:

$$N_y = (1 + \bar{Y}_{11})(1 + \bar{Y}_{22}) - \bar{Y}_{12}\bar{Y}_{21} \tag{49}$$

Converting Y-Matrix to Z-Matrix

The complex Y-matrix is converted to the complex Z-matrix using the formulas:

$$\begin{aligned} Z_{11} &= \frac{Y_{22}}{D_y} \\ Z_{12} &= \frac{-Y_{12}}{D_y} \\ Z_{21} &= \frac{-Y_{21}}{D_y} \\ Z_{22} &= \frac{Y_{11}}{D_y} \end{aligned} \tag{50}$$

with $D_y = Y_{11}Y_{22} - Y_{12}Y_{21}$.

Gains, Amplifier Stability, and Unilateralization

Small-Signal Current Gain

The short-circuit small-signal current gain h_{21} of a transistor is given by:

$$h_{21} = \left|\frac{Y_{21}}{Y_{11}}\right| \tag{51}$$

Amplifier Stability

The Rollett stability factor K is computed from the S-parameters using the formula [6][7]:

$$K = \frac{1 - |S_{11}|^2 - |S_{22}|^2 + |\Delta|^2}{2|S_{12} \cdot S_{21}|} \tag{52}$$

where:

$$|\Delta| = |S_{11} \cdot S_{22} - S_{12} \cdot S_{21}| \tag{53}$$

The necessary and sufficient conditions for unconditional stability for an amplifier are [6]:

$$K > 1 \tag{54}$$

$$|\Delta| < 1 \tag{55}$$

Unconditional stability indicates conjugate matching between output and input loads. For $K < 1$, an amplifier is conditionally stable or potentially unstable and must be stabilized.

Maximum Stable Gain and Maximum Available Gain

The maximum stable gain (MSG) G_{ms} of a two-port network is given by [7][8]:

$$G_{\mathrm{ms}} = \left|\frac{S_{21}}{S_{12}}\right| \tag{56}$$

The maximum available gain (MAG) G_{ma} depends on the stability of the two-port network. For an unconditionally stable two-port network, that is, if both $K > 1$ and $|\Delta| < 1$:

$$G_{\mathrm{ma}}(K > 1, |\Delta| < 1) = G_{\mathrm{ms}} \cdot (\mathrm{K} - \sqrt{\mathrm{K}^2 - 1}) \tag{57}$$

For $K \leq 1$ or $|\Delta| > 1$, MAG is set to MSG [9]:

$$G_{\mathrm{ma}}(K \leq 1, |\Delta| > 1) = G_{\mathrm{ms}} \tag{58}$$

Unilateral Amplifier Design

Mason's unilateral gain (MUG) U is computed from the S-parameters using the formula [8]:

$$U = \frac{\left|\frac{S_{21}}{S_{12}} - 1\right|^2}{2K\left|\frac{S_{21}}{S_{12}}\right| - 2 \cdot \Re(\frac{S_{21}}{S_{12}})} \tag{59}$$

where $\Re(z)$ denotes the real part of the complex number z.

The unilateral figure of merit U_{f} is given by [6]:

$$U_{\mathrm{f}} = \frac{|S_{12}||S_{21}||S_{22}||S_{11}|}{(1 - |S_{11}|^2)(1 - |S_{22}|^2)} \tag{60}$$

For a unilateral amplifier design approach, U_{f} must be as small as possible.

Converting Gain Units to Decibels

The short-circuit current gain, $|h_{21}|$, can be expressed in units of decibel (dB) using:

$$|h_{21}|[\mathrm{dB}] = 20\log|h_{21}| \tag{61}$$

The power gain, P, is expressed in units of dB using:

$$P[\mathrm{dB}] = 10\log P \tag{62}$$

Transistor Figures of Merit

ft and fmax

The frequency dependency of the magnitude of the current gain $|h_{21}|$ is given by [2]:

$$|h_{21}| \approx \frac{\beta}{\sqrt{1+\left(\frac{f}{f_\beta}\right)^2}} \tag{63}$$

where β is the current gain at low frequency, and f_β is the β cutoff frequency or the 3 dB frequency.

The short-circuit current gain cutoff frequency or the cutoff frequency f_t is defined as the frequency at which $|h_{21}| = 1$ (unit gain point):

$$f_t \equiv f(|h_{21}| = 1 = 0[\mathrm{dB}]) \tag{64}$$

f_t is related to f_β:

$$f_t = \beta f_\beta \tag{65}$$

Eq. 63 shows that for $f \ll f_\beta$ (low frequencies):

$$|h_{21}| \approx \beta \tag{66}$$

and for $f \gg f_\beta$ (high frequencies):

$$|h_{21}| \approx \frac{f_t}{f} \tag{67}$$

Converting the unit of $|h_{21}|$ to dB (using Eq. 61), Eq. 67 can be written as:

$$|h_{21}|[\mathrm{dB}] = 20\log f_t - 20\log f \tag{68}$$

Therefore, the $|h_{21}|$ (in units of dB) versus $\log f$ curve (current gain curve) is flat at low frequencies, reduces by 3 dB at f_β, and falls off linearly with a slope of –20 dB/decade with increasing frequencies.

Let $(\log f_0, |h_{21,0}|[\mathrm{dB}])$ be a high frequency point at which the –20 dB/decade slope is fully established. From Eq. 68:

$$20\log\frac{f_t}{f_0} = |h_{21,0}|[\mathrm{dB}] \tag{69}$$

or:

$$f_t = f_0 \cdot 10^{|h_{21,0}|[\mathrm{dB}]/20} \tag{70}$$

Therefore, Eq. 68 implies that f_t can also be determined by linear extrapolation from a high frequency point $(\log f_0, |h_{21,0}|[\mathrm{dB}])$ on the current gain curve, using Eq. 70 [9].

The frequency dependency of MUG or MAG at high frequencies is given by [9]:

$$G \approx \frac{f_{\max}^2}{f^2} \tag{71}$$

where G is either MUG or MAG, and $f_{\max}$ is the maximum frequency of oscillation. $f_{\max}$ can be extracted using either MUG or MAG [9].

$f_{\max}$ is defined as the frequency at which $U = 1$ (unit gain point):

$$f_{\max} \equiv f(U = 1 = 0[\mathrm{dB}]) \tag{72}$$

or the frequency at which $G_{\mathrm{ma}} = 1$:

$$f_{\max} \equiv f(G_{\mathrm{ma}} = 1 = 0[\mathrm{dB}]) \tag{73}$$

$f_{\max}$ is the maximum frequency at which power gain can be extracted from an amplifier. It is also the maximum frequency of oscillation of an oscillator made from an amplifier with power gain. If $U > 1$, the transistor is active and $f_{\max}$ is extracted. If $U \le 1$, the transistor is passive and $f_{\max}$ is not extracted [10].

Similar to f_t, f_{max} also can be determined by linear extrapolation from a point $(\log f_0, U_0[\mathrm{dB}])$ on the power gain curve (U in units of dB versus $\log f$) with a slope of –20 dB/decade, using [2]:

$$f_{max} = f_0 \cdot 10^{(U_0[\mathrm{dB}])/20} \tag{74}$$

A similar equation is used to extract f_{max} from a G_{ma} versus f curve:

$$f_{max} = f_0 \cdot 10^{(G_{ma,0}[\mathrm{dB}])/20} \tag{75}$$

Extraction Methods for ft and fmax

f_t and f_{max} are extracted from the corresponding gain curves by the following RF extraction library procedures:

- `rfx::GetFt` extracts f_t from the $|h_{21}|$ versus frequency curve.
- `rfx::GetFmax` extracts f_{max} from the U versus frequency curve, or the G_{ma} versus frequency curve. For brevity, either of these power gains is denoted by G.

Both procedures use three different methods of extraction: `unit-gain-point`, `extract-at-dBPoint`, and `extract-at-frequency`. The last two methods are extrapolation methods and assume the ideal frequency dependency of gain (flat at low frequencies, and falling off linearly at higher frequencies with a slope of –20 dB/decade). None of these methods checks the validity of these assumptions.

The extraction methods are (see Figure 121 on page 455):

(a) The *unit gain point method* uses the definition of f_t (Eq. 64) and f_{max} (Eq. 72 or Eq. 73). It searches directly for the unit gain point but may give inappropriate results if the gain curves deviate from the –20 dB/decade slope near the unit gain point.

(b) The *extract at dB point method* looks for the gain point $((\log f_0, |h_{21,0}|[\mathrm{dB}])$ or $(\log f_0, G_0[\mathrm{dB}]))$ where the gain ($|h_{21}|$ or G) has fallen by a certain number of decibels from its value at the start of the gain curve. This difference in decibels is called the *dB point* and is specified using the keyword `-parameter` in units of dB. Assuming a –20 dB/decade slope, the gain point $((\log f_0, |h_{21,0}|[\mathrm{dB}])$ or $(\log f_0, G_0[\mathrm{dB}]))$ is used to compute f_t using Eq. 70 or f_{max} using Eq. 74 or Eq. 75. This method may give inappropriate results if the –20 dB/decade slope is not fully established at the gain point. Often, the results can be improved by adjusting the dB point.

(c) The *extract at frequency method* is the same as method (b), but the frequency corresponding to the unit gain point is extrapolated from a specified frequency f_0. The frequency f_0 is specified using the keyword `-parameter`. The corresponding gain ($|h_{21,0}|[\mathrm{dB}]$ or $G_0[\mathrm{dB}]$) is computed, and f_t or f_{max} is computed using Eq. 70 or Eq. 74 and Eq. 75. This method may

give inappropriate results if the –20 dB/decade slope is not fully established at this gain point. The optimal frequency for this method may depend on the control bias or current, and may be different for G and $|h_{21}|$.

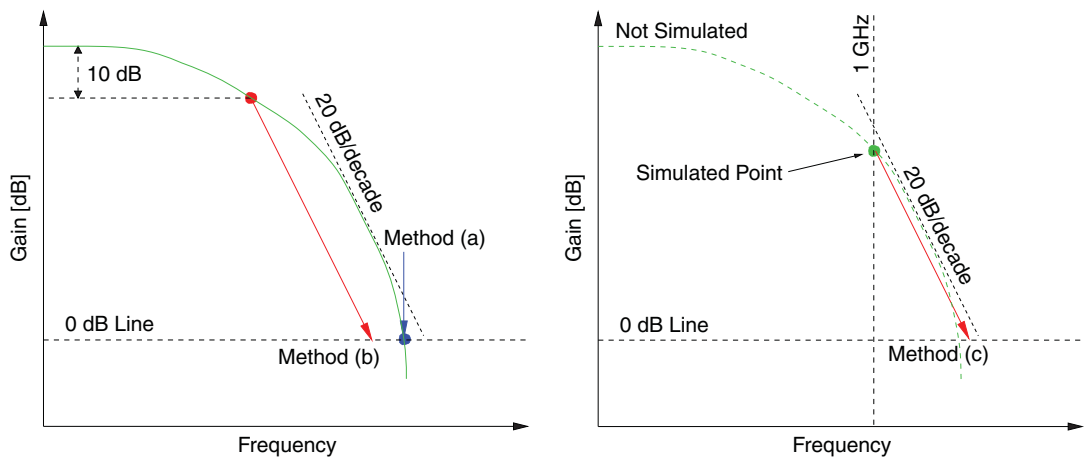


Figure 121 Different extraction methods and the circumstances under which they may return inappropriate results

Unfortunately, no single extraction method of f_t and f_{max} is appropriate under all circumstances. Technically, method (a), the direct search for the unit gain point, should be the most appropriate method. However, at high frequencies, additional parasitic elements may become dominant and alter the –20 dB/decade slope near the unit gain point. Furthermore, in experiments, it is sometimes difficult to trace the gain curve to high-enough frequencies to see the unit gain point directly. Therefore, extrapolation of experimental data to the unit gain point is common. In this case, the experiment may not 'see' the altered slope due to the parasitics (also some parasitics may not be included in the simulation). For a comparison with experimental results, therefore, method (b) or method (c) may be better.

The extraction methods assume that, for each value of the control bias or current, a full frequency sweep is performed. Ideally, this sweep should start at a frequency where the gain is flat (low-frequency regime) and should end beyond the unit gain point. If the frequency sweep does not go beyond the unit gain point, method (a) sets the value of f_t to zero. If the frequency sweep does not start in the flat region, methods (b) and (c) still return a (nonzero) value. However, you must ensure that, at the selected dB point (for method (b)) or the frequency point (for method (c)), the –20 dB/decade slope is established.

The transition between the flat low-frequency region of the gain curves to the –20 dB/decade slope at higher frequencies can be wide. Sometimes, a clear –20 dB/decade slope is never reached. In this case, methods (b) and (c) may give incorrect results (often, the results can be improved by adjusting the dB point used for the extrapolation).

The simulation of a full frequency sweep in Sentaurus Device can be time consuming, especially for large structures and if many equations are solved. If it is known beforehand that, at a given frequency, the slope of the gain curves is –20 dB/decade, it is sufficient to simulate the small-signal response at this single frequency only and to apply method (c). However, the band of frequencies for which the –20 dB/decade slope assumption holds true can be very narrow and can depend on the bias conditions.

This discussion shows that using solely one method may give inappropriate results. Therefore, it is recommended to always use all three methods concurrently. If the f_t or f_{max} curves for all three methods agree well, the results can be trusted with a high level of confidence. If the results are very different, most likely, the form of the gain curve prevents a meaningful automatic extraction of f_t and f_{max}. In this case, it is suggested to examine the underlying gain curves and the slope of the gain curves. In most such cases, it is clear that the assumptions on which the extractions are based are not fulfilled. That is, the gain curve does not fall off with a clean –20 dB/decade slope at high frequencies.

Cutoff Frequency for Stability

The cutoff frequency for stability f_{K1} is defined as the frequency point at which $K = 1$ (the boundary between the unconditionally stable and conditionally stable region):

$$f_{K1} \equiv f(K = 1) \tag{76}$$

Noise Figure of a Linear Two-Port Network

The noise factor F of a linear two-port network is defined as the signal-to-noise ratio at the input port divided by the signal-to-noise ratio at the output port.

In the RF extraction library, the y-parameters of the noisy two-port network and the PSDs of the admittance representation are used to compute the PSDs of the chain representation, which are then used to compute the noise figure and the noise parameters of the two-port network.

As discussed in Power Spectral Densities on page 444, in the chain representation, the noisy two-port network consists of two noise sources, v_n and i_n, placed at the input port. The two-port is driven by a sinusoidal source of either internal admittance Y_s (source admittance) or internal impedance Z_s (source impedance). The noise figure NF (Eq. 96) of the noisy two-port network is the noise factor F expressed in units of dB (Eq. 94). It is determined by the source admittance and the noise parameters of the two-port [9]. The noise parameters are:

- Minimum noise figure, NF_{min} (Eq. 95)
- Equivalent noise resistance R_n of the noise voltage source (Eq. 78)
- Optimum source admittance Y_{opt} (Eq. 89 and Eq. 90)

The optimum source admittance is the value of the source admittance at which the noise figure has its minimum value $NF_{\min}$. R_n determines the sensitivity of the noise figure to deviations from Y_{opt}.

An alternative set of noise parameters is [11]:

- Minimum noise figure, $NF_{\min}$ (Eq. 95)
- Equivalent noise conductance G_n of the noise voltage source (Eq. 84)
- Optimum source impedance Z_{opt} (Eq. 92)

The above admittances and impedances can be normalized by dividing by the characteristic impedance Z_o, and these values are called normalized admittances and impedances (r_n, y_{opt}, g_n, and z_{opt}).

The equations used to compute the noise figure, the noise parameters, and various other quantities using the `rfx::NoiseFigure` procedure (see rfx::NoiseFigure on page 486) are discussed here [2][11][12][13][14].

The spectral density of the equivalent input noise voltage source S_{v_n} is computed using:

$$S_{v_n} = \frac{S_{\mathrm{I}}^{22}}{|Y_{21}|} \tag{77}$$

The equivalent noise resistance R_n of the noise voltage source is computed using:

$$R_n = \frac{S_{v_n}}{4k_{\mathrm{B}}T_o} \tag{78}$$

where:

- k_{B} is the Boltzmann constant.
- $T_o = 290\,\mathrm{K}$ is the standard noise temperature.

The normalized equivalent noise resistance r_n is computed using:

$$r_n = \frac{R_n}{Z_o} \tag{79}$$

The equivalent noise conductance G_u and the normalized equivalent noise conductance g_u of the uncorrelated noise current component are given by:

$$G_u = \frac{1}{4k_{\mathrm{B}}T_o}\left(S_{\mathrm{I}}^{11} - -\frac{\left|S_{\mathrm{I}}^{12}\right|^2}{S_{\mathrm{I}}^{22}}\right) \tag{80}$$

$$g_u = \frac{G_u}{Z_o} \tag{81}$$

The correlation admittance Y_{cor} is given by:

$$Y_{\text{cor}} = G_{\text{cor}} + jB_{\text{cor}} = Y_{11} - Y_{21}\frac{S_{\text{I}}^{12}}{S_{\text{I}}^{22}} \tag{82}$$

where:

- G_{cor} is the correlation conductance.
- B_{cor} is the correlation susceptance.

The spectral density of the equivalent input noise current source S_{i_n} is given by:

$$S_{i_n} = 4k_{\text{B}}T_o(\left|Y_{\text{cor}}\right|^2 R_n + G_u) \tag{83}$$

The equivalent noise conductance G_n and the normalized equivalent noise conductance g_n of the input noise current source are computed using:

$$\begin{aligned} G_n &= \frac{S_{i_n}}{4k_{\text{B}}T_o} \\ g_n &= \frac{G_n}{Z_o} \end{aligned} \tag{84}$$

The cross-correlation spectral densities $S_{v_n\bar{i}_n}$ and $S_{i_n\bar{v}_n}$ of the equivalent input noise voltage and noise current sources are given by:

$$\begin{aligned} S_{v_n\bar{i}_n} &= \overline{Y_{\text{cor}}}S_{v_n} \\ S_{i_n\bar{v}_n} &= \overline{S_{v_n\bar{i}_n}} \end{aligned} \tag{85}$$

The noise correlation coefficient between the equivalent input noise voltage and noise current source is computed using:

$$C_{i_n v_n} = \frac{S_{i_n\bar{v}_n}}{\sqrt{S_{i_n}S_{v_n}}} \tag{86}$$

The source admittance and the source impedance are defined as:

$$\begin{aligned} Y_{\text{s}} &= G_{\text{s}} + jB_{\text{s}} \\ Z_{\text{s}} &= R_{\text{s}} + jX_{\text{s}} = \frac{1}{Y_{\text{s}}} \end{aligned} \tag{87}$$

where:

- G_s is the source conductance.
- B_s is the source susceptance.
- R_s is the source resistance.
- X_s is the source reactance.

The optimum source admittance Y_{opt} is defined as:

$$Y_{opt} = G_{opt} + jB_{opt} \tag{88}$$

The optimum source conductance G_{opt} is computed using:

$$G_{opt} = \sqrt{\frac{G_u + R_n G_{cor}^2}{R_n}} \tag{89}$$

The optimum source susceptance B_{opt} is computed using:

$$B_{opt} = -B_{cor} \tag{90}$$

The normalized optimum source admittance y_{opt} is computed using:

$$y_{opt} = \frac{Y_{opt}}{Z_o} \tag{91}$$

The optimum source impedance Z_{opt} and the normalized optimum source impedance z_{opt} are defined as:

$$\begin{aligned} Z_{opt} &= R_{opt} + jX_{opt} = \frac{1}{Y_{opt}} \\ z_{opt} &= \frac{Z_{opt}}{Z_o} \end{aligned} \tag{92}$$

The minimum noise factor F_{min} is given by:

$$F_{min} = 1 + 2R_n(G_{cor} + G_{opt}) \tag{93}$$

and the noise factor F is computed using:

$$F = F_{min} + \frac{R_n}{G_s}[(G_s - G_{opt})^2 + (B_s - B_{opt})^2] \tag{94}$$

The minimum noise figure $NF_{\min}$ [dB] is computed using:

$$NF_{\min} = 10 \log_{10} F_{\min} \tag{95}$$

The noise figure NF [dB] is computed using:

$$NF = 10 \log_{10} F \tag{96}$$

rfx Namespace Variables

Many RF extraction library procedures use the variables summarized in Table 31. These variables are created by the `rfx::Load` procedure.

NOTE If there is a named noise specification in the Sentaurus Device command file, this name is prefixed to the name of the PSD Tcl array, for example, `<name>_rfx::SV`.

Table 31 rfx namespace variables

Variable name	Description
`rfx::AC`	AC array (see A-Matrix, C-Matrix, and Y-Matrix on page 442).
`rfx::Y`	Y-matrix (see A-Matrix, C-Matrix, and Y-Matrix on page 442).
`rfx::nfreq`	Number of frequencies.
`rfx::freq`	List of frequencies.
`rfx::i_freqstart`	Index of the first element in the list of frequencies, `rfx::freq`.
`rfx::i_freqend`	Index of the last element in the list of frequencies, `rfx::freq`.
`rfx::nbias`	Number of bias points.
`rfx::bias`	List of bias points.
`rfx::i_biasstart`	Index of the first element in the list of bias points, `rfx::bias`.
`rfx::i_biasend`	Index of the last element in the list of bias points, `rfx::bias`.
`rfx::z0`	Characteristic impedance in units of Ω. Default: $50\,\Omega$.
`rfx::Zs`	Source impedance [Ω] seen by the two-port network. Default: `[$rfx::z0 0]`
`rfx::T0`	Standard noise temperature.
Noise or power spectral density arrays	
`rfx::SV`	NVSD matrix (S_V^{11}, S_V^{12}, S_V^{21}, and S_V^{22}).
`rfx::SI`	NISD matrix (S_I^{11}, S_I^{12}, S_I^{21}, and S_I^{22}).

Table 31 rfx namespace variables

Variable name	Description
Partial noise spectral density arrays	
`rfx::SVee`	Electron NVSD matrix.
`rfx::SVhh`	Hole NVSD matrix.
`rfx::SIee`	Electron NISD matrix.
`rfx::SIhh`	Hole NISD matrix.
`rfx::SVeeDiff`	Electron NVSD matrix due to diffusion LNS.
`rfx::SVhhDiff`	Hole NVSD matrix due to diffusion LNS.
`rfx::SVeeMonoGR`	Electron NVSD matrix due to monopolar generation–recombination (GR) LNS.
`rfx::SVhhMonoGR`	Hole NVSD matrix due to monopolar GR LNS.
`rfx::SVeeFlickerGR`	Electron NVSD matrix due to flicker GR LNS.
`rfx::SVhhFlickerGR`	Hole NVSD matrix due to flicker GR LNS.

Characteristic Impedance and Source Impedance

The RF extraction library uses the characteristic impedance Z_o to:

- Convert Y-parameters to S-parameters (see Eq. 47, p. 449 and rfx::Y2S on page 494).
- Compute the normalized values of various noise parameter–related admittances and impedances (see Noise Figure of a Linear Two-Port Network on page 456).

The RF extraction library also uses the source impedance Z_s to compute the noise figure (see Eq. 96, p. 460 and rfx::NoiseFigure on page 486).

In the RF extraction library, Z_o and Z_s are represented by the `rfx::z0` and `rfx::Zs` variables, respectively (see Table 31 on page 460).

Z_o defaults to $50\,\Omega$. To change the characteristic impedance to $100\,\Omega$, for example, use the following command *after* loading the RF extraction library:

```
set rfx::z0 100.0
```

Z_s defaults to $50 + j0\,\Omega$. To change the source impedance to $100 + j0\,\Omega$, for example, use the following command *after* loading the RF extraction library:

```
set rfx::Zs [list 100.0 0.0]
```

rfx::CreateDataset

Creates a Sentaurus Visual dataset corresponding to an RF matrix or a PSD matrix as a function of frequency or bias.

Syntax

```
rfx::CreateDataset -dataset <dataName>
   [-rfmatrix "AC" | "Y" | "H" | "Z" | "S" | "SV" | "SI"] [-dB 0 | 10 | 20]
   [-noisename <string>] [-xaxis "frequency" | "bias"]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-dataset <dataName>`	Name of a Sentaurus Visual dataset. The dataset contains variables for all the RF parameters in the RF matrix specified using the keyword `-rfmatrix`, as a function of frequency for each bias point (`-xaxis "frequency"`) or bias for each frequency (`-xaxis "bias"`). For `-xaxis "frequency"`, the variables are created for all the bias point indices (`0` to `rfx::i_biasend`). For `-xaxis "bias"`, the variables are created for all the frequency point indices (`0` to `rfx::i_freqend`). For example, for `-rfmatrix "Y"` and `-xaxis "frequency"`, the dataset contains the Y-parameters as a function of frequency for each bias point. Therefore, the variables `"bias_<i> y<ij>_Re"` and `"bias_<i> frequency"` are created. Here, `i` is the bias point index and `ij` refers to the port numbers (11, 12, 21, 22). If `-dB 10` (or `20`) is specified, 10dB (or 20dB) is appended to the name of the variable corresponding to the absolute value of the RF parameter. The variables are summarized in Table 32 on page 463. Similar variables are created for other RF parameters. For `-rfmatrix "SV"` or `-rfmatrix "SI"`, variables for all the power spectral densities are created. These are summarized in Table 33 on page 464. In addition, the variables for all of the Y-parameters are created. (String, no default)
`-rfmatrix "AC" | "Y" | "H" | "Z" | "S" | "SV" | "SI"`	Name of the RF or PSD matrix. (String, default: `"AC"`)
`-dB 0 | 10 | 20`	Decibel level for computing the absolute value of an RF parameter. For `-dB 0`, the absolute value is computed on linear scale. Default: `0`
`-noisename <string>`	Name of the noise specification. Required only for `-rfmatrix "SV"` or `-rfmatrix "SI"`, and in the case when a named noise specification was used to perform noise analysis. (String, default: `""`)
`-xaxis "frequency" | "bias"`	Selects the x-axis as either frequency or bias. Selects the sorting order. For example, for `-xaxis "frequency"`, the data is created using a loop with frequency as the *inner variable* and bias as the *outer variable*. (String, default: `"frequency"`)

`-info 0	1	2	3`	Sets local info level. Default: `0`
`-help 0	1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
rfx::Load -dataset "ACPLT" -file AC_des.plt -port1 1 -port2 2 -biasport "v(1)"

# Create dataset for Y-parameters as a function of frequency
rfx::CreateDataset -dataset "2port_Y_freq" -xaxis "frequency" -rfmatrix "Y"

# Create dataset for NVSD as a function of frequency
rfx::CreateDataset -dataset "2port_PSD_freq" -xaxis "frequency" -rfmatrix "SV"

# Create dataset for NISD as a function of bias and for a named noise
specification
rfx::CreateDataset -dataset "2port_PSD_bias" -xaxis "bias" -rfmatrix "SI" \
   -noisename diff
```

Table 32 Dataset variable names for -xaxis "frequency" and -rfmatrix "Y"

Dataset variable name	Description
`bias_<i> frequency`	List of frequencies.
`bias_<i> y<ij>_Re`	List of the real parts of the Y-parameter, Y_{ij}.
`bias_<i> y<ij>_Im`	List of the imaginary parts of the Y-parameter, Y_{ij}.
`bias_<i> y<ij>_Abs`	List of the absolute values of the Y-parameter, Y_{ij}. This variable is created if the keyword `-dB` is not specified or `-dB 0` is specified.
`bias_<i> y<ij>_Abs_10dB`	List of the absolute values of the Y-parameter, Y_{ij}, on a 10 dB scale. This variable is created only if `-dB 10` is specified.
`bias_<i> y<ij>_Abs_20dB`	List of the absolute values of the Y-parameter, Y_{ij}, on a 20 dB scale. This variable is created only if `-dB 20` is specified.
`bias_<i> y<ij>_Phase`	List of the phases of the Y-parameter, Y_{ij}.
Here, `ij` refers to the port numbers (11, 12, 21, 22); `i` is the bias point index; `iv` varies from `rfx::i_biasstart` to `rfx::i_biasend`. These variables are created for all the Y-parameters: Y_{11}, Y_{12}, Y_{21}, and Y_{22}.	

Table 33 Dataset variable names for -xaxis "frequency" and -rfmatrix "SV"

Dataset variable name	Description
`bias_<i> frequency`	List of frequencies.
`bias_<i> sv<ij>`	List of NVSD autocorrelation coefficients (S_V^{ij}, $i = j$).
`bias_<i> sv<ij>_Re`	List of the real parts of the NVSD cross-correlation coefficients (S_V^{ij}, $i \neq j$).
`bias_<i> sv<ij>_Im`	List of the imaginary parts of the NVSD cross-correlation coefficients (S_V^{ij}, $i \neq j$).
`bias_<i> sv<ij>_Abs`	List of the absolute values of the NVSD cross-correlation coefficients, (S_V^{ij}, $i \neq j$). This variable is created if the keyword `-dB` is not specified or `-dB 0` is specified.
`bias_<i> sv<ij>_Phase`	List of the phases of the cross-correlation coefficients, (S_V^{ij}, $i \neq j$).
Here, `ij` refers to the port numbers (11, 12, 21, 22); `i` is the bias point index; `iv` varies from `rfx::i_biasstart` to `rfx::i_biasend`. These variables are created for all of the coefficients of the S_V-matrix: S_V^{11}, S_V^{12}, S_V^{21}, and S_V^{22}.	

NOTE Table 33 lists the variables corresponding to S_V^{ij}. Similar variables are created for S_I^{ij}. For the partial noise spectral densities that describe the contribution of specific noise sources, the name of the specific noise source is included in parentheses, for example, `svij(ee)`, `svij(eeMonoGR)`, and `svij_Re(eeMonoGR)`. If there is a named noise specification, this name is prefixed to the name of the variable. For example, if the name of the noise specification is `diff`, examples of variable names are `diff_svij(ee)`, `diff_svij(eeDiff)`, and `diff_svij_Re(eeDiff)`.

NOTE Table 34 on page 465 lists examples of PSD matrix coefficients and the corresponding names of PSD variables in the AC data file, the PSD Tcl array name and element, and the name of the corresponding dataset variables.

Table 34 PSD data in AC data file, PSD Tcl array element, and dataset variables

Coefficient of PSD matrix	PSD variable in AC data file	PSD Tcl array element	Dataset variables
NVSD matrix S_V			
$S_V^{11}(f, v)$	`S_V(1)`	`rfx::SV(0,1,1,$if, $iv)`	`sv11`

$S_V^{12}(f, v)$	–	`rfx::SV(0,1,2,$if, $iv)`
		`rfx::SV(1,1,2,$if, $iv)`
			`sv12_Re`
			`sv12_Im`
			`sv12_Abs`
			`sv12_Phase`

$S_V^{21}(f, v)$	`ReS_VXV(2,1)`
	`ImS_VXV(2,1)`
		`rfx::SV(0,2,1,$if, $iv)`
		`rfx::SV(1,2,1,$if, $iv)`
			`sv21_Re`
			`sv21_Im`
			`sv21_Abs`
			`sv21_Phase`

$S_V^{22}(f, v)$	`S_V(2)`	`rfx::SV(2,2)`	`sv22`
The matrix $S_{V,n}^{GR}$ with coefficients corresponding to electron NVSD due to monopolar GR LNS (a partial PSD)			
$S_{V,n}^{GR,11}(f, v)$	`S_V_eeMonoGR(1)`	`rfx::SVeeMonoGR(0,1,1,$if,$iv)`	`sv11(eeMonoGR)`

$S_{V,n}^{GR,12}(f, v)$	–	`rfx::SVeeMonoGR(0,1,2,$if,$iv)`
		`rfx::SVeeMonoGR(1,1,2,$if,$iv)`
			`sv12_Re(eeMonoGR)`
			`sv12_Im(eeMonoGR)`
			`sv12_Abs(eeMonoGR)`
			`sv12_Phase(eeMonoGR)`

$S_{V,n}^{GR,21}(f, v)$	`ReS_VXV_eeMonoGR (2,1)`
	`ImS_VXV_eeMonoGR (2,1)`
		`rfx::SVeeMonoGR(0,2,1,$if,$iv)`
		`rfx::SVeeMonoGR(1,2,1,$if,$iv)`
			`sv21_Re(eeMonoGR)`
			`sv21_Im(eeMonoGR)`
			`sv21_Abs(eeMonoGR)`
			`sv21_Phase(eeMonoGR)`

$S_{V,n}^{GR,22}(f, v)$	`S_V_eeMonoGR(2)`	`rfx::SVeeMonoGR(0,2,2,$if,$iv)`	`sv22(eeMonoGR)`

rfx::Export

Exports the AC array, or the Y-, h-, Z-, or S-matrix, to a CSV file.

Syntax

```
rfx::Export -rfmatrix "AC" | "Y" | "H" | "Z" | "S"
   [-file <fileName>] [-xaxis "frequency" | "bias"]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-rfmatrix "AC" | "Y" | "H" | "Z" | "S"`	Name of the RF matrix. (String, no default)
`-file <fileName>`	Name of the CSV file. Default: `"rfmatrix.csv"`
`-xaxis "frequency" | "bias"`	Specifies either the frequency or the bias as the axis. Selects the sorting order. For example, for `-xaxis "frequency"`, the data is printed using a loop with frequency as the *inner variable* and bias as the *outer variable*. Default: `"frequency"`
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Create AC array and Y-matrix.
rfx::Load -dataset "ACPLT" -file "AC_des.plt" -port1 1 -port2 2 \
   -biasport "v(1)"
rfx::Export -file S_f.csv -rfmatrix "S" -xaxis "frequency"
```

NOTE The CSV file can be loaded into any spreadsheet application. It contains a header that lists the number of bias and frequency points as well as the value of the first and last bias and frequency points. The header is followed by a table, which contains the frequency, the bias, and the real and imaginary parts of the elements of the RF matrix. Two versions of the CSV file can be written. One is sorted by frequencies and the other is sorted by bias points. The keyword `-xaxis` specifies whether the parameters should be sorted by frequency first (`-xaxis "frequency"`), with bias being the secondary parameter, or by bias first (`-xaxis "bias"`) with frequency being the secondary parameter. For example, the CSV file `S_freq.csv` generated by the `rfx::Export` command in the above example contains the following (for `-xaxis "bias"`, the first two columns are reversed):

```
# of bias pts.  : 21,  first bias: -0.5,  last bias: 0.5
# of frequencies: 13,  first freq: 1e+08,  last freq: 1e+12
bias,freq,S11_Re,S11_Im,S12_Re,S12_Im,S21_Re,S21_Im,S22_Re,S22_Im
-0.5,1e+08,0.999,-0.002,-2.570e-06,0.0002,-1.297,0.002,0.933,
-6.750e-04
-0.5,2.15e+08,0.999,-0.004,-1.34e-06,0.0005,-1.297,0.0052,0.93,-0.001
-0.5,4.64e+08,0.999,-0.010,4.336e-06,0.001,-1.297,0.012,0.933,-0.003
...
```

rfx::GetFK1

Computes the cutoff frequency for stability f_{K1} (see Cutoff Frequency for Stability on page 456) at all bias points from the Rollett stability factor versus frequency curves. Creates the corresponding datasets if the keyword `-dataset` is specified.

NOTE If f_{K1} is not found, the procedure returns 0.

Syntax

```
rfx::GetFK1 -out <array_name> [-dataset <dataName>] [-xscale "lin" | "log"]
   [-scale <r>] [-target <r>] [-occurrence <i>] [-info 0 | 1 | 2 | 3]
   [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements fK1 and bias. The values of the fK1 element and the bias element are lists of f_{K1} and bias, respectively. (Array name, no default)
`-dataset <dataName>`	Name of Sentaurus Visual dataset containing the variables `bias` and `fK1`. The variable `bias` contains a list of bias values, and the variable `fK1` contains a list of cutoff frequencies for stability. These variables can be used to plot a f_{K1} versus bias curve. The dataset is created only if this keyword is specified. (String, no default)
`-xscale "lin" | "log"`	Specifies whether the values on the x-axis are linearly or logarithmically distributed. Default: `"log"`
`-scale <r>`	Computed f_{K1} is divided by this scaling factor. Use to convert, for example, the f_{K1} value to GHz. (Real number, default: `1.0`)
`-target <r>`	Selects the value of K that should be looked for. (Real number, default: `1.0`)
`-occurrence <i>`	Specifies the *n*-th interpolated f_{K1} value to be extracted. Use this if multiple frequencies have the same K-value (specified using the keyword `-target`) at a bias point. (Integer, default: `1`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Create Y-matrix and corresponding dataset (AC and Y-matrix).
rfx::Load -dataset "ACPLT" -file "AC_des.plt" -port1 1 -port2 2 \
   -biasport "v(1)"

# # Compute fK1 versus bias and corresponding dataset.
rfx::GetFK1 -out FK -dataset "fK_bias" -xscale "log" -occurrence 1 -scale 1e9
puts "Bias Points= $FK(bias)"
puts "Cutoff frequencies for stability \[GHz\]= $FK(fK1)"

#-> Bias Points= -0.5 -0.45 -0.4 ...
#-> Cutoff frequencies for stability [GHz]= 135.380 133.130 130.748 ...
```

rfx::GetFmax

Computes the maximum frequency of oscillation f_{max} at all bias points from the power gain (MUG or MAG) versus frequency curves at each bias point. Creates the corresponding datasets if the keywords `dataset` and `slopedataset` are specified.

NOTE This procedure can compute f_{max} using three different methods (see Extraction Methods for ft and fmax on page 454). If f_{max} is not found, or the power gain is less than or equal to 1 (device is passive), the procedure returns 0.

Syntax

```
rfx::GetFmax -out <array_name> -parameter <r> [-method <string>]
   [-dataset <dataName>] [-slopedataset <dataName>]
   [-powergain "MUG" | "MAG"] [-xscale "lin" | "log"] [-scale <r>]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements fmax and bias. The values of the fmax element and the bias element are lists of f_{max} and bias, respectively. (Array name, no default)
`-parameter <r>`	The dB point for method `"b"`, specified in units of dB, or the frequency point in units of Hz for method `"c"`. This is a mandatory argument if method `"b"` or `"c"` is used. (Real number, no default)
`-method <string>`	Specifies the method to use for computing f_{max} (see Extraction Methods for ft and fmax on page 454): • `"a"` or `"unit-gain-point"` extracts f_{max} as the frequency at which power gain equals one. • `"b"` or `"extract-at-dbpoint"` extracts f_{max} by extrapolating the power gain from the point at which it has fallen by a certain number of decibels (called the dB point) from its initial value. • `"c"` or `"extract-at-frequency"` is the same as `"b"`, but the power gain is extrapolated from the specified frequency point. (String, default: `"a"`)
`-dataset <dataName>`	Name of Sentaurus Visual dataset containing the variables `bias` and `fmax`. The variable `bias` contains a list of bias values, and the variable `fmax` contains a list of f_{max} values. These variables can be used to plot a f_{max} versus bias curve. The dataset is created only if `-dataset` is specified. (String, no default)

`-slopedataset <dataName>`	Name of Sentaurus Visual dataset containing the variables "`bias_<i> frequency`" and "`bias_<i> dMUG`" (`-powergain "MUG"`) or "`bias_<i> dMAG`" (`-powergain "MAG"`) corresponding to the bias point `bias_<i>`. `i` is the bias point index. These variables are created for all the bias point indices (`0` to `rfx::i_biasend`). The variable `frequency` contains a list of frequency values, and the variables `dMUG` and `dMAG` contain a list of derivatives of MUG and MAG, respectively as a function of frequency. The unit of the power gain derivatives is dB/decade. These variables are used to plot the power gain derivative versus frequency curve for various bias points. The dataset is created only if `-slopedataset` is specified. (String, no default)
`-powergain "MUG" | "MAG"`	Selects the power gain used for extracting $f_{\max}$. (String, default: "`MUG`")
`-xscale "lin" | "log"`	Specifies whether the values on the x-axis are linearly or logarithmically distributed. Default: `"log"`
`-scale <r>`	Computed $f_{\max}$ is divided by this scaling factor. Use to convert, for example, the $f_{\max}$ value to GHz. (Real number, default: `1.0`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Create Y-matrix and corresponding dataset (AC array and Y-matrix).
rfx::Load -dataset "ACPLT" -file "AC_des.plt" -port1 1 -port2 2 \
   -biasport "v(1)"

# Compute fmax Vs bias. Create datasets of dMUG Vs frequency and fmax Vs bias.
rfx::GetFmax -out Fmax0 -method "unit-gain-point" -scale 1e9 -xscale "log" \
   -dataset "fmax0_bias" -slopedataset "MUG_slope_freq"

puts "Max frequency of oscillation= $Fmax0(fmax)"
puts "Bias Points= $Fmax0(bias)"

#-> Max frequency of oscillation= 195.560 190.733 185.724 ...
#-> Bias Points= -0.5 -0.45 -0.4 ...
```

rfx::GetFt

Computes the cutoff frequency f_t at all bias points from the current gain $|h_{21}|$ versus frequency curves at each bias point. Creates the corresponding datasets if the keywords `dataset` and `slopedataset` are specified.

NOTE This procedure can compute f_t using three different methods (see Extraction Methods for ft and fmax on page 454). If f_t is not found or $|h_{21}| \leq 1$, the procedure returns 0.

Syntax

```
rfx::GetFt -out <array_name> -parameter <r> [-method <string>]
   [-dataset <dataName>] [-slopedataset <dataName>] [-xscale "lin" | "log"]
   [-scale <r>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements ft and bias. The values of the ft element and the bias element are lists of f_t and bias, respectively. (Array name, no default)
`-parameter <r>`	The dB point for method `"b"`, specified in units of dB, or the frequency point in units of Hz for method `"c"`. This is a mandatory argument if method `"b"` or `"c"` is used. (Real number, no default)
`-method <string>`	Specifies the method to use for computing f_t (see Extraction Methods for ft and fmax on page 454): • `"a"` or `"unit-gain-point"` extracts f_t as the frequency at which $\|h_{21}\| = 0\,\mathrm{dB}$. • `"b"` or `"extract-at-dbpoint"` extracts f_t by extrapolating $\|h_{21}\|$ from the point at which $\|h_{21}\|$ has fallen by a certain number of decibels (called the dB point) from its initial value. • `"c"` or `"extract-at-frequency"` is the same as `"b"`, but $\|h_{21}\|$ is extrapolated from the specified frequency point. (String, default: `"a"`)
`-dataset <dataName>`	Name of Sentaurus Visual dataset containing the variables `bias` and `ft`. The variable `bias` contains a list of bias values, and the variable `ft` contains a list of f_t values. These variables can be used to plot a f_t versus bias curve. The dataset is created only if `-dataset` is specified. (String, no default)

`-slopedataset <dataName>`	Name of Sentaurus Visual dataset containing the variables "`bias_<i> frequency`" and "`bias_<i> dh21`" corresponding to the bias point `bias_<i>`. `i` is the bias point index. These variables are created for all the bias point indices (`0` to `rfx::i_biasend`). The variable frequency contains a list of frequency values, and the variable `dh21` contains a list of the derivatives of $\lvert h_{21} \rvert$ as a function of frequency. The unit of `dh21` is dB/decade. It can be used to plot the derivatives of $\lvert h_{21} \rvert$ versus frequency curve at various bias points. The slope dataset is created only if `-slopedataset` is specified. (String, no default)
`-xscale "lin" | "log"`	Specifies whether the values on the x-axis are linearly or logarithmically distributed. Default: `"log"`
`-scale <r>`	Computed f_t is divided by this scaling factor. Use to convert, for example, the f_t value to GHz. (Real number, default: `1.0`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Create Y-matrix and corresponding dataset (AC and Y-matrix)
rfx::Load -dataset "ACPLT" -file "AC_des.plt" -port1 1 -port2 2 \
   -biasport "v(1)"
# Create dataset of h-parameters versus frequency
rfx::CreateDataset -dataset 2port_H_freq -xaxis "frequency" \
   -rfmatrix "H" -dB 20
# Compute ft Vs bias. Create datasets of dh21 Vs frequency and
rfx::GetFt -out Ft0 -method "unit-gain-point" -scale 1e9 -xscale "log" \
   -dataset "ft0_bias" -slopedataset "h21_slope_freq"
puts "Cutoff frequencies= $Ft0(ft)"
puts "Bias Points= $Ft0(bias)"
```

```
#-> Cutoff frequencies= 67.985 66.819 65.594 ...
#-> Bias Points= -0.5 -0.45 -0.4 ...
```

rfx::GetNearestIndex

Finds the index of the entry in an ordered numeric list that is closest to the given target. For example, this procedure can find the index of a frequency or bias point closest to a frequency or bias point of interest (see Tcl Arrays rfx::AC and rfx::Y on page 443).

NOTE If the target is outside the range of values in the numeric list, this procedure returns the index of the first element or the last element in the numeric list. Therefore, if the frequency or bias point is outside the range of frequency or bias values, this procedure returns the index of the first element or the last element in the list of frequency or bias values.

Syntax

```
rfx::GetNearestIndex -out <var_name> -target <r> -list <list_of_r>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <var_name>`	Variable name to store the index.
`-target <r>`	Target value. (Real number, no default)
`-list <list_of_r>`	An ordered numeric list. The list can be in ascending or descending order. (List of real numbers, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
rfx::Load -dataset "ACPLT" -file "AC_des.plt" -port1 1 -port2 2 \
   -biasport "v(1)"
```

```
rfx::GetNearestIndex -out i_bias -target 0.025 -list $rfx::bias
puts "Nearest bias point index is $i_bias"
set BiasPoint [lindex $rfx::bias $i_bias]
puts "Corresponding bias point is $BiasPoint"
```

```
rfx::GetNearestIndex -out i_freq -target 1e9 -list $rfx::freq
puts "Nearest frequency index is $i_freq"
set Frequency [lindex $rfx::freq $i_freq]
puts "Corresponding frequency is $Frequency"
```

```
#-> Nearest bias point index is 11
#-> Corresponding bias point is 0.05
#-> Nearest frequency index is 3
#-> Corresponding frequency is 1e+09
```

rfx::GetNoiseFigure

Computes the noise figure, the noise parameters, the input-referred spectral densities, and several other parameters as a function of frequency or bias:

- Noise figure NF (using Eq. 96, p. 460) and minimum noise figure $NF_{\min}$ (using Eq. 95, p. 460)
- Noise factor F (using Eq. 94, p. 459) and minimum noise factor $F_{\min}$ (using Eq. 93, p. 459)
- Equivalent noise resistance R_n (using Eq. 78, p. 457) and conductance G_n (using Eq. 84, p. 458)
- Optimum source admittance Y_{opt} (using Eq. 89 and Eq. 90, p. 459) and impedance Z_{opt} (using Eq. 92, p. 459)
- Equivalent noise conductance G_u (using Eq. 80, p. 457) and correlation admittance Y_{cor} (using Eq. 82, p. 458)
- Normalized quantities r_n, g_n, y_{opt}, z_{opt}, and g_u
- Input-referred spectral densities S_{v_n} (using Eq. 77, p. 457), $S_{v_n \bar{i}_n}$ (using Eq. 85, p. 458), $S_{i_n \bar{v}_n}$ (using Eq. 85), and S_{i_n} (using Eq. 83, p. 458)
- Noise correlation coefficients C_{i_n} (using Eq. 41, p. 446) and $C_{i_n v_n}$ (using Eq. 86, p. 458)

NOTE This procedure uses the `rfx` namespace variables `rfx::zo` and `rfx::Zs` (see Characteristic Impedance and Source Impedance on page 461).

Syntax

```
rfx::GetNoiseFigure -out <array_name> [-xaxis "frequency" | "bias"]
   (-target <r> | -index <i>) [-dataset <dataName>]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements summarized in Table 35 on page 476, which also summarizes the values of these elements. The index also contains the element freq (for `-xaxis "frequency"`) or bias (for `-xaxis "bias"`). (Array name, no default)

`-xaxis "frequency" | "bias"`	Specifies either the frequency or bias as the axis. Default: `"frequency"`
`-target <r>`	Bias point (for `-xaxis "frequency"`) or frequency point (for `-xaxis "bias"`). Specify only one of the keywords `-target` or `-index`. (Real number, no default)
`-index <i>`	Index of the bias point (for `-xaxis "frequency"`) or frequency point (for `-xaxis "bias"`). Specify only one of the keywords `-target` or `-index`. (Integer, no default)
`-dataset <dataName>`	Name of Sentaurus Visual dataset containing the variables summarized in Table 35. The dataset also contains the variable frequency (for `-xaxis "frequency"`) or bias (for `-xaxis "bias"`). (String, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Table 35 Elements of array index, dataset variable names, and their values

Elements of array index	Dataset variable name	Value
NF, F, NFmin, Fmin	`NF, F, NFmin, Fmin`	Noise figure, noise factor, minimum noise figure, and minimum noise factor.
Rn, Gn, rn, gn	`Rn, Gn, rn, gn`	Equivalent noise resistance R_n and conductance G_n, and their normalized values r_n and g_n.
Gopt, Bopt, Ropt, Xopt, gopt, bopt, ropt, xopt	`Gopt, Bopt, Ropt, Xopt, gopt, bopt, ropt, xopt`	Optimum source conductance, susceptance, resistance, and reactance, and their normalized values.
Gcor, Bcor	`Gcor, Bcor`	Correlation conductance and susceptance.
Gu, gu	`Gu, gu`	Equivalent conductance G_u and its normalized values g_u.
ReCi, ImCi, AbsCi, PhaseCi	`Ci_Re, Ci_Im, Ci_Abs, Ci_Phase`	Real and imaginary parts, absolute value, and phase of C_{i_n}.
Sv, ReSvi, ImSvi, ReSiv, ImSiv, Si	`Sv, Svi_Re, Svi_Im, Siv_Re, Siv_Im, Si`	S_{v_n}, real and imaginary parts of $S_{v_n \bar{i}_n}$, real and imaginary parts of $S_{i_n \bar{v}_n}$, and S_{i_n}.
ReCiv, ImCiv, AbsCiv, PhaseCiv	`Civ_Re, Civ_Im, Civ_Abs, Civ_Phase`	Real and imaginary parts, absolute value, and phase of $C_{i_n v_n}$.

Returns

None.

Example

```
# Create Y-matrix and PSD matrices
rfx::Load -dataset "ACPLT" -file "noise_des.plt" -port1 1 -port2 2 \
   -biasport "v(1)"
# Compute noise parameters, noise figure and input-referred spectral densities
# at 0 bias, as a function of frequency
rfx::GetNoiseFigure -out NFparam -dataset "NF_freq" -xaxis "frequency" \
   -target 0.0

puts "frequency= $NFparam(freq)"
puts "NF= $NFparam(NF)"
puts "NFmin= $NFparam(NFmin)"
puts "Rn= $NFparam(Rn)"
puts "Gopt= $NFparam(Gopt)"
puts "Bopt= $NFparam(Bopt)"
puts "Sv= $NFparam(Sv)"
puts "Si= $NFparam(Si)"
puts "ReSvi= $NFparam(ReSvi)"
puts "ImSvi= $NFparam(ImSvi)"

#-> frequency= 0.1 0.215 0.464 ...
#-> NF= 113.505 113.505 113.505...
#-> NFmin= 5.211e-07 1.129e-06 2.475e-06 ...
#-> Rn= 1.120e13 1.120e13 1.120e13 ...
#-> Gopt= 5.463e-21 1.177e-20 2.535e-20 ...
#-> Bopt= -2.198e-16 -4.736e-16 -1.020e-15 ...
#-> Sv= 1.794e-07 1.794e-07 1.794e-07 ...
#-> Si= 8.673e-39 4.026e-38 1.868e-37 ...
#-> ReSvi= -1.087e-36 -5.047e-36 -2.342e-35 ...
#-> ImSvi= -3.945e-23 -8.500e-23 -1.831e-22 ...
```

rfx::GetParsAtPoint

Accesses the RF parameters of an RF matrix at a given bias and frequency point.

Syntax

```
rfx::GetParsAtPoint -out <array_name> -rfmatrix "AC" | "Y" | "H" | "Z" | "S"
   -biaspoint <r> -freqpoint <r> [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements bias, freq, 11, 12, 21, and 22. The values of the bias element and the freq element are the bias and the frequency point, respectively. The 11, 12, 21, and 22 elements are the complex RF parameters (list containing real and imaginary parts). (Array name, no default)
`-rfmatrix "AC" | "Y" | "H" | "Z" | "S"`	Name of the RF matrix. (String, no default)
`-biaspoint <r>`	Bias point. (Real number, no default)
`-freqpoint <r>`	Frequency point. (Real number, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Create AC array and Y-matrix
rfx::Load -dataset "ACPLT" -file "AC_des.plt" -port1 1 -port2 2 \
   -biasport "v(1)"
set BiasPoint [lindex $rfx::bias 0]
set Frequency [lindex $rfx::freq 0]
rfx::GetParsAtPoint -out ReImData -rfmatrix "AC" -biaspoint $BiasPoint \
   -freqpoint $Frequency
puts "Bias Point= $ReImData(bias)"
puts "Freq Point= $ReImData(freq)"
puts "ac11= $ReImData(11)"
puts "ac12= $ReImData(12)"
puts "ac21= $ReImData(21)"
puts "ac22= $ReImData(22)"
```

```
#-> Bias Point= -0.5
#-> Freq Point= 1e+08
#-> ac11= 1.21582e-09 1.32657e-15
#-> ac12= 1.21693e-09 -1.75088e-16
#-> ac21= 0.000536849 -6.06722e-16
#-> ac22= 2.76303e-05 3.42598e-16
```

rfx::GetPowerGain

Computes the following as a function of frequency (at a fixed bias point) or bias (at a fixed frequency point):

- Rollett stability factor (using Eq. 52) and stability condition delta (using Eq. 53)
- MSG (using Eq. 56) (linear scale as well as 10 dB scale)
- MAG (using Eq. 57 and Eq. 58) (linear scale as well as 10 dB scale)
- MUG (using Eq. 59) (linear scale as well as 10 dB scale)
- Unilateral figure of merit (using Eq. 60)

NOTE If the denominator in Eq. 52, Eq. 56, Eq. 59, or Eq. 60 is 0, the procedure returns a value of 10^{20}.

NOTE If MUG, MSG, or MAG is 0, the procedure returns a value of 10^{-20}.

Syntax

```
rfx::GetPowerGain -out <array_name> [-xaxis "frequency" | "bias"]
   (-target <r> | -index <i>) [-dataset <dataName>]
   [-powergain "all" | "MUG" | "MSG"| "MAG"]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements K, delta, and freq (for `-xaxis "frequency"`) or bias (for `-xaxis "bias"`). The values of the K, delta, freq, and bias elements are the Rollett stability factor, the stability condition delta, the frequency, and the bias, respectively. In addition, for `-powergain "all"`, the index contains the elements MUG, MUG_dB, MSG, MSG_dB, MAG, MAG_dB, and U. These are MUG (linear scale), MUG (10 dB scale), MSG (linear scale), MSG (10 dB scale), MAG (linear scale), MAG (10 dB scale), and U_f, respectively. For `-powergain "MUG"`, only the MUG, MUG_dB, and U elements are created. For `-powergain "MSG"`, only the MSG and MSG_dB elements are created. For `-powergain "MAG"`, only the MAG and MAG_dB elements are created. (Array name, no default)
`-xaxis "frequency" | "bias"`	Specifies either the frequency or bias as the axis. Default: `"frequency"`
`-target <r>`	Bias point (for `-xaxis "frequency"`) or frequency point (for `-xaxis "bias"`). Specify only one of the keywords `-target` or `-index`. (Real number, no default)

`-index <i>`	Index of the bias point (for `-xaxis "frequency"`) or frequency point (for `-xaxis "bias"`). Specify only one of the keywords `-target` or `-index`. (Integer, no default)
`-dataset <dataName>`	Name of Sentaurus Visual dataset containing the variables K, delta, and frequency (for `-xaxis "frequency"`) or bias (for `-xaxis "bias"`). In addition, for `-powergain "all"`, the variables MSG, MSG_dB, MAG, MAG_dB, MUG, MUG_dB, and U are created. For `-powergain "MUG"`, the variables MUG, MUG_dB, and U are created. For `-powergain "MSG"`, the variables MSG and MSG_dB are created. For `-powergain "MAG"`, the variables MAG and MAG_dB are created. (String, no default)
`-powergain "all" | "MUG" | "MSG"| "MAG"`	Specifies the power gains to compute: • For `-powergain "all"`, MSG, MAG, MUG, and U are computed. • For `-powergain "MUG"`, only MUG and U are computed. • For `-powergain "MSG"`, only MSG is computed. • For `-powergain "MAG"`, only MAG is computed. The power gains are computed on both the linear scale and 10 dB scale. (String, default: `"all"`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# Create AC array and Y-matrix
rfx::Load -file "AC_des.plt" -port1 1 -port2 2 -biasport "v(1)"
# Compute stability factor and power gain at 0 bias, as a function of frequency
rfx::GetPowerGain -out gain -dataset "Pgain_freq" -xaxis "frequency" \
   -target 0.0
puts "frequency= $gain(freq)"
puts "K= $gain(K)"
puts "delta= $gain(delta)"
puts "MSG= $gain(MSG)"
puts "MSG_dB= $gain(MSG_dB)"
puts "MAG= $gain(MAG)"
puts "MAG_dB= $gain(MAG_dB)"
puts "MUG= $gain(MUG)"
puts "MUG_dB= $gain(MUG_dB)"
puts "U= $gain(U)"
```

```
#-> frequency= 1e+08 2.154e+08 4.641e+08 ...
#-> K= -0.125 -0.057 -0.023 ...
#-> delta= 0.996 0.996 0.996 ...
#-> MUG= -6090.93 -6082.684 -6044.973 ...
#-> MUG_dB= 37.846 37.840 37.813
#-> MSG= 3297.908 1543.462 717.697
#-> MSG_dB= 35.182 31.884 28.559
#-> MAG= 3297.908 1543.462 717.697
#-> MAG_dB= 35.182 31.884 28.559
#-> U= 27.354 56.375 104.099 ...
```

rfx::Load

Loads a Sentaurus Device AC data file and creates a Sentaurus Visual dataset. It also creates the Tcl arrays, `rfx::AC` and `rfx::Y` (see Tcl Arrays rfx::AC and rfx::Y on page 443), along with several `rfx` namespace variables summarized in Table 31 on page 460. The Tcl arrays for the power spectral densities are created only if the AC data file contains PSD data (see Power Spectral Density Tcl Arrays on page 447).

Syntax

```
rfx::Load -file <fileName>
   [-biasport <stringValue>] [-biassportsign +1 | -1] [-dataset <dataName>]
   [-devicewidth <r>] [-port1 <integer | stringValue>]
   [-port2 <integer | stringValue>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-file <fileName>`	Name of the Sentaurus Device AC data file. (String, no default)
`-biasport <stringValue>`	Name of biased port. For example, `"v(1)"` for voltage on port or node `1`, or `"i(vc,2)"` for current flowing out of the voltage source `vc` at port or node `2`. (String, default: `"v(1)"`)
`-biassportsign +1 | -1`	Sign of the bias port. Used to reverse the polarity of the bias. Default: `+1`
`-dataset <dataName>`	Name of the created dataset. It contains the data from the Sentaurus Device AC data file. (String, default: `"ACPLT"`)
`-devicewidth <r>`	Device width multiplier (device width in the z-direction, L_z ; see Device Width Scaling for 2D Structures on page 448) in µm . (Real number, default: `1.0`)
`-port1 <integer | stringValue>`	Name of the input port of the two-port network. The port name must agree with the node names defined in the Sentaurus Device `System` section. (Integer or string, default: `1`)
`-port2 <integer | stringValue>`	Name of the output port of the two-port network. The port name must agree with the node names defined in the Sentaurus Device `System` section. (Integer or string, default: `2`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
rfx::Load -dataset "ACPLT" -file AC_des.plt -port1 1 -port2 2 -biasport "v(1)"
puts "Number of bias points= $rfx::nbias"
puts "Bias points= $rfx::bias"
puts "Number of frequencies= $rfx::nfreq"
puts "Frequencies= $rfx::freq"
puts "printing the Y-matrix"
set biases [get_variable_data v(1) -dataset ACPLT]
foreach P1 {1 2} {
   foreach P1 {1 2} {
      set if 0                              ;# Frequency index
      set iv 0                              ;# Bias point index
      foreach Bias $biases {
         puts "Frequency= [lindex $rfx::freq $if]"
         puts "Bias point= [lindex $rfx::bias $iv]"
         puts "Y(0,$P1,$P2,$if,$iv)= $rfx::Y(0,$P1,$P2,$if,$iv)"
         puts "Y(1,$P1,$P2,$if,$iv)= $rfx::Y(0,$P1,$P2,$if,$iv)"
         if { $iv < [expr $rfx::nbias-1] } {
            incr iv
         } elseif { $if < [expr $rfx::nfreq-1] } {
            incr if
            set iv 0
         }
      }
   }
}
#-> Number of bias points= 3
#-> Bias points= -0.5 -0.45 -0.4
#-> Number of frequencies= 4
#-> Frequencies= 1e+08 4.64159e+08 2.15443e+09 1e+10
#-> printing the Y-matrix
#-> Frequency= 1e+08
#-> Bias point= -0.5
#-> Y(0,1,1,0,0)= 1.21582e-09
#-> Y(1,1,1,0,0)= 8.33508513295e-07
#-> Frequency= 1e+08
#-> Bias point= -0.45
#-> Y(0,1,1,0,1)= 1.55703e-09
#-> Y(1,1,1,0,1)= 8.38993734068e-07
...
```

NOTE It is assumed that the Sentaurus Device AC data file contains one or more frequency sweeps with a voltage bias or current bias as the control variable.

NOTE The keyword `-biasport` defines which port is biased and whether the biasing is a voltage or a current condition. For a current condition, the syntax for `biasport` is more complex. For example, if the name of the current source is `vc` and the name of the port is `2`, `-biasport` is specified as `-biasport "i(vc,2)"`.

In this case, however, Sentaurus Device must also be instructed to include the current at this node through this device in the Sentaurus Device AC data file. This is performed in the `System` section of the Sentaurus Device input file with, for example:

```
System {
   HBT hbt (base=1 collector=2 emitter=0)
   Vsource_pset vb ( 1 0 ){ dc = 0 }
   Vsource_pset vc ( 2 0 ){ dc = 0 }
   ACPlot(v(1) v(2) i(vb 1) i(vc 2))
}
```

NOTE Internally, the arrays `rfx::AC` and `rfx::Y` use the port numbers 1 and 2 corresponding to `port1` and `port2` as array indices (see Tcl Arrays rfx::AC and rfx::Y on page 443). This convention is also valid for the PSD Tcl arrays.

rfx::NoiseFigure

Computes the noise figure, the noise parameters, the power spectral densities, and other parameters at a fixed frequency and bias point:

- Noise figure NF (using Eq. 96, p. 460) and minimum noise figure $NF_{\min}$ (using Eq. 95, p. 460)
- Noise factor F (using Eq. 94, p. 459) and minimum noise factor $F_{\min}$ (using Eq. 93, p. 459)
- Equivalent noise resistance R_n (using Eq. 78, p. 457) and conductance G_n (using Eq. 84, p. 458)
- Optimum source admittance Y_{opt} (using Eq. 89 and Eq. 90, p. 459) and impedance Z_{opt} (using Eq. 92, p. 459)
- Equivalent noise conductance G_u (using Eq. 80, p. 457) and correlation admittance Y_{cor} (using Eq. 82, p. 458)
- Normalized quantities r_n, g_n, y_{opt}, z_{opt}, and g_u
- Input-referred spectral densities S_{v_n} (using Eq. 77, p. 457), $S_{v_n \bar{i}_n}$ (using Eq. 85, p. 458), $S_{i_n \bar{v}_n}$ (using Eq. 85), and S_{i_n} (using Eq. 83, p. 458)
- Noise correlation coefficients C_{i_n} (using Eq. 41, p. 446) and $C_{i_n v_n}$ (using Eq. 86, p. 458)

NOTE This procedure uses the `rfx` namespace variables `rfx::zo` and `rfx::Zs` (see Characteristic Impedance and Source Impedance on page 461).

Syntax

```
rfx::NoiseFigure SI11 SI12 SI22 Y11 Y21
```

Argument	Description
`SI11 SI12 SI22`	The coefficients S_{I}^{11}, S_{I}^{12}, and S_{I}^{22} of the complex NISD matrix for a fixed frequency and bias point in the form of three lists. Each list contains the real and imaginary parts. (List of real numbers, no default)
`Y11 Y21`	The complex elements Y_{11} and Y_{21} of the Y-matrix for a fixed frequency and bias point in the form of two lists. Each list contains the real and imaginary parts. (List of real numbers, no default)

Returns

An array having one string-valued index. The index contains elements summarized in Table 35 on page 476, which also summarizes the values of these elements.

Example

```
set SI11 [list 8.24593987e-23 0.0]
set SI12 [list -3.91285654e-23 -1.08922612e-23]
set SI22 [list 4.57789473e-23 0.0]
set Y11  [list 0.00613324387 0.0170027533]
set Y21  [list -0.00506706586 -0.0060744537]
rfx::NoiseFigure $SI11 $SI12 $SI22 $Y11 $Y21
puts "NF= $NFparameters(NF)"
puts "NFmin= $NFparameters(NFmin)"
puts "Rn= $NFparameters(Rn)"
puts "Gopt= [lindex $NFparameters(Yopt) 0]"
puts "Bopt= [lindex $NFparameters(Yopt) 1]"
puts "Sv= $NFparameters(Sv)"
puts "Si= $NFparameters(Si)"
puts "ReSvi= [lindex $NFparameters(Svi) 0]"
puts "ImSvi= [lindex $NFparameters(Svi) 1]"
```

```
#-> NF= 4.20978496
#-> NFmin= 3.18609968
#-> Rn= 45.680068
#-> Gopt= 0.00860243885
#-> Bopt= -0.0106051299
#-> Sv= 7.3159526e-19
#-> Si= 1.36421106e-22
#-> ReSvi= 2.37591689e-21
#-> ImSvi= -7.75866277e-21
```

rfx::PolarBackdrop

Creates a ruled background on which RF parameters in polar coordinates are plotted. Creates two families of curves: a set of concentric circles and a set of angular lines.

Syntax

```
rfx::PolarBackdrop -plot <plotName> -r <list_of_r> -phi <list_of_r>
   [-dataset <dataName>] [-color <stringValue>] [-linestyle <stringValue>]
   [-linewidth <r>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-plot <plotName>`	Name of the polar plot. (String, no default)
`-r <list_of_r>`	A list of increasing radial values. (List of real numbers, no default)
`-phi <list_of_r>`	A list of increasing angular values. (List of real numbers, no default)
`-dataset <dataName>`	Name of the dataset used to create the polar background. (String, default: `"PolarBackdrop"`)
`-color <stringValue>`	Sets the color of the curves of the polar background. (String, default: `"black"`)
`-linestyle <stringValue>`	Sets the style of the curve lines of the polar background. (String, default: `"dash"`)
`-linewidth <r>`	Sets the line width of the curve lines of the polar background. (Real number, default: `2.0`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Rs [list 0.25 0.5 0.75 1.0]
set Phis [list 0 30 60 90 120 150]
rfx::PolarBackdrop -plot Plot_Polar -r $Rs -phi $Phis
```

rfx::PowerGain

Computes the following at a fixed bias and frequency point:

- Rollett stability factor (using Eq. 52) and stability condition delta (using Eq. 53)
- MSG (using Eq. 56) (linear scale)
- MAG (using Eq. 57 and Eq. 58) (linear scale)
- MUG (using Eq. 59) (linear scale)
- Unilateral figure of merit (using Eq. 60)

Syntax

```
rfx::PowerGain S11 S12 S21 S22
```

Argument	Description
`S11 S12 S21 S22`	The complex S-matrix for a fixed frequency and bias point in the form of four lists. Each list contains the real and imaginary parts of an element S_{ij} of the S-matrix. (List of real numbers, no default)

Returns

The Rollett stability factor, the stability condition delta, MSG, MAG, MUG, and U_f at a fixed frequency and bias point in the form of a list.

Example

```
set S11 [list 0.999999877967 -8.36457327936e-05]
set S12 [list -1.20947562066e-07 1.09859319808e-05]
set S21 [list -0.0536108305232 4.08872504539e-05]
set S22 [list 0.997240784695 -2.17611978854e-05]
set Pgain [rfx::PowerGain $S11 $S12 $S21 $S22]
puts "K= [lindex $Pgain 0]"
puts "delta= [lindex $Pgain 1]"
puts "MSG= [lindex $Pgain 2]"
puts "MAG= [lindex $Pgain 3]"
puts "MUG= [lindex $Pgain 4]
puts "U= [lindex $Pgain 5]""
```

```
#-> K= -0.009
#-> delta= 0.997
#-> MSG= 4879.658
#-> MAG= 4879.658
#-> MUG= -116259.175
#-> U= 449.598
```

rfx::RFCList

Accesses a slice of the Y-, h-, Z-, S-matrix, or PSD matrices, typically, to generate a curve of an RF parameter or PSD as a function of bias or frequency.

Syntax

```
rfx::RFCList -out <array_name> -rfparameter <string> -index <i>
   [-noisename <string>]
   [-noisesource "ee" | "hh" | "eeDiff" | "hhDiff" | "eeMonoGR" | "hhMonoGR" |
      "eeFlickerGR" | "hhFlickerGR"]
   [-xaxis "frequency" | "bias"] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are lists of the real part and imaginary part, respectively, of the RF parameter. (Array name, no default)
`-rfparameter <string>`	Parameter identifier. Use, for example, the notation `h21` for the h-matrix element h_{21} . For PSDs, use `sv<ij>` or `si<ij>`. Here, `ij` refers to the port numbers (11, 12, 21, 22). In addition, for partial noise spectral density, use the `-noisesource` keyword. (String, no default)
`-index <i>`	Selects the index of the slice. (Integer, no default)
`-noisename <string>`	Name of the noise specification. Required only for `-rfparameter sv<ij>` or `si<ij>`, and when a named noise specification was used to perform noise analysis. (String, default: `""`)
`-noisesource "ee" | "hh" | "eeDiff" | "hhDiff" | "eeMonoGR" | "hhMonoGR" | "eeFlickerGR" | "hhFlickerGR"`	Name of the noise source. Only used for accessing the matrices of partial PSDs. For `-rfparameter si<ij>`, the only allowed values of this keyword are `"ee"` and `"hh"`. (String, no default)
`-xaxis "frequency" | "bias"`	Specifies either the frequency or bias as the axis. Default: `"frequency"`
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
# For v=-0.5, return real and imaginary parts of z11 as a function of
# frequency.
rfx::Load -dataset ACPLT -file AC_des.plt -port1 1 -port2 2 -biasport "v(1)"
rfx::GetNearestIndex -out i_bias -target -0.5 -list $rfx::bias
puts "Nearest bias point index is $i_bias"
set BiasPoint [lindex $rfx::bias $i_bias]
puts "Corresponding bias point is $BiasPoint"
rfx::RFCList -out ReImz11 -rfparameter "z11" -xaxis "frequency" -index $i_bias
puts "Re(ReImz11)= $ReImz11(Re)"
puts "Im(ReImz11)= $ReImz11(Im)"
#-> Nearest bias point index is 0
#-> Corresponding bias point is -0.5
#-> Re(ReImz11)= -482.364 1501.418 1928.740 ...
#-> Re(ReImz11)= -482.364 1501.418 1928.740 ...
```

```
# Accessing a slice of an NVSD
rfx::RFCList -out ReImsv11 -rfparameter "sv11" -xaxis "frequency" \
   -index $i_bias
# Accessing a slice of an NISD
rfx::RFCList -out ReImsi11 -rfparameter "si11" -xaxis "frequency"
   -index $i_bias
# Accessing a slice of a partial PSD
rfx::RFCList -out ReImsvee11 -rfparameter "sv11" -noisesource "ee" \
   -xaxis "frequency" -index $i_bias
```

rfx::SmithBackdrop

Creates a Smith chart–ruled background on which RF parameters are plotted. Creates two families of curves: the normalized resistance circles and the normalized reactance (capacitive or inductive) arcs.

Syntax

```
rfx::SmithBackdrop -plot <plotName> -r <list_of_r> -x <list_of_r>
   [-dataset <dataName>] [-color <stringValue>] [-linestyle <stringValue>]
   [-linewidth <r>] [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-plot <plotName>`	Name of the Smith chart. (String, no default)
`-r <list_of_r>`	A list of normalized resistance values. The list must contain positive values, which monotonically increase. (List of real numbers, no default)
`-x <list_of_r>`	A list of normalized reactance values. The list must contain nonzero positive values, which monotonically increase. (List of real numbers, no default)
`-dataset <dataName>`	Name of dataset used to create the Smith chart background. (String, default: `"SmithBackdrop"`)
`-color <stringValue>`	Sets the color of the curves of the Smith chart background. (String, default: `"black"`)
`-linestyle <stringValue>`	Sets the style of the curve lines of the Smith chart background. (String, default: `"dash"`)
`-linewidth <r>`	Sets the line width of the curve lines of the Smith chart background. (Real, default: `2.0`)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Rs [list 0 0.3333 1.0 3.0]
set Xs [list 0.268 0.575 1 1.73 3.75]
rfx::SmithBackdrop -plot Plot_Smith -r $Rs -x $Xs
```

rfx::Y2H

Converts Y-parameters to h-parameters at a fixed frequency and bias point using Eq. 46, p. 449.

Syntax

```
rfx::Y2H Y11 Y12 Y21 Y22
```

Argument	Description
`Y11 Y12 Y21 Y22`	The complex Y-matrix for a fixed frequency and bias point in the form of four lists. Each list contains the real and imaginary parts of an element Y_{ij} of the Y-matrix. (List of real numbers, no default)

Returns

The complex h-matrix at the fixed frequency and bias point in the form of four lists. Each list contains the real and imaginary parts of an element h_{ij} of the h-matrix.

Example

```
set Y11 [list 1.21582e-09 8.33508513295e-07]
set Y12 [list 1.21693e-09 -1.10011034906e-07]
set Y21 [list 0.000536849 -3.81214675594e-07]
set Y22 [list 2.76303e-05 2.15260671987e-07]
set H [rfx::Y2H $Y11 $Y12 $Y21 $Y22]
puts "h11= [lindex $H 0]"
puts "h12= [lindex $H 1]"
puts "h21= [lindex $H 2]"
puts "h22= [lindex $H 3]"
```

```
#-> h11= 1750.04122429 -1199745.24113
#-> h12= 0.131983085922 0.00165252982249
#-> h21= 0.482147388329 -644.082700094
#-> h22= 9.84859176627e-05 1.05210576538e-06
```

rfx::Y2S

Converts Y-parameters to S-parameters at a fixed frequency and bias point using Eq. 47, p. 449. The S-parameters are computed using a characteristic impedance value that defaults to $50\,\Omega$.

NOTE This procedure uses the variable `rfx::z0` (see Characteristic Impedance and Source Impedance on page 461).

Syntax

```
rfx::Y2S Y11 Y12 Y21 Y22
```

Argument	Description
`Y11 Y12 Y21 Y22`	The complex Y-matrix for a fixed frequency and bias point in the form of four lists. Each list contains the real and imaginary parts of an element Y_{ij} of the Y-matrix. (List of real numbers, no default)

Returns

The complex S-matrix in the form of four lists at a fixed frequency and bias point. Each list contains the real and imaginary parts of an element S_{ij} of the S-matrix.

Example

```
set rfx::z0 100.0
set Y11 [list 1.21582e-09 8.33508513295e-07]
set Y12 [list 1.21693e-09 -1.10011034906e-07]
set Y21 [list 0.000536849 -3.81214675594e-07]
set Y22 [list 2.76303e-05 2.15260671987e-07]
set S [rfx::Y2S $Y11 $Y12 $Y21 $Y22]
puts "S11= [lindex $S 0]"
puts "S12= [lindex $S 1]"
puts "S21= [lindex $S 2]"
puts "S22= [lindex $S 3]"
```

```
#-> S11= 0.999999754906 -0.000167879603832
#-> S12= -2.40402553357e-07 2.19416045779e-05
#-> S21= -0.107073930082 8.73191407279e-05
#-> S22= 0.994489177684 -4.39899046514e-05
```

rfx::Y2Z

Converts Y-parameters to Z-parameters at a fixed frequency and bias point using Eq. 50, p. 450.

Syntax

```
rfx::Y2Z Y11 Y12 Y21 Y22
```

Argument	Description
`Y11 Y12 Y21 Y22`	The complex Y-matrix for a fixed frequency and bias point in the form of four lists. Each list contains the real and imaginary parts of an element Y_{ij} of the Y-matrix. (List of real numbers, no default)

Returns

The complex Z-matrix in the form of four lists at a fixed frequency and bias point. Each list contains the real and imaginary parts of an element Z_{ij} of the Z-matrix.

Example

```
set Y11 [list 1.21582e-09 8.33508513295e-07]
set Y12 [list 1.21693e-09 -1.10011034906e-07]
set Y21 [list 0.000536849 -3.81214675594e-07]
set Y22 [list 2.76303e-05 2.15260671987e-07]
set Z [rfx::Y2Z $Y11 $Y12 $Y21 $Y22]
puts "Z11= [lindex $Z 0]"
puts "Z12= [lindex $Z 1]"
puts "Z21= [lindex $Z 2]"
puts "Z22= [lindex $Z 3]"
```

```
#-> Z11= -482.364647889 -336580.472711
#-> Z12= 1340.14771043 2.46281596039
#-> Z21= 64960.8791939 6539151.68447
#-> Z22= 10152.5772867 -108.457994303
```

Complex Arithmetic Support

This section describes procedures for performing complex arithmetic. For most procedures, the RF extraction library contains two versions of a procedure: scalar and vectorial. The name of the scalar version of a procedure ends with *c*, and the name of the corresponding vectorial version ends with *v*.

The scalar version operates on a single complex number or two complex numbers. A complex number is specified using a Tcl list containing the real and imaginary parts of the complex number. For example, the absolute value of the complex number, $z = 4 + 3i$ can be computed using the procedure `rfx::Abs_c` as follows:

```
set z [list 4 3]
puts [rfx::Abs_c $z]
#-> 5.0
```

The vectorial version operates on either a single list of complex numbers or two lists of complex numbers. The list of complex numbers is specified using arrays that have one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the list of complex numbers. For example, the absolute values of the complex numbers $z_1 = 2 + 3i$ and $z_2 = -1 + i$ can be computed using the procedure `rfx::Abs_v` as follows:

```
set Z(Re) [list 2 -1]
set Z(Im) [list 3 1]
rfx::Abs_v -out absvals -z Z
puts "abs values = $absvals"
#-> abs values = 3.606 1.414
```

All the procedures except `rfx::Polar2Cart_c` and `rfx::Polar2Cart_v` operate on complex numbers specified in Cartesian coordinates. The `rfx::Polar2Cart` procedures operate on a complex number or a list of complex numbers specified in polar coordinates, which are represented by a Tcl list or an array, respectively. For example, the complex number $\sqrt{2}\angle 45°$ is specified using:

```
set z [list 1.414 45]
```

rfx::Abs_c

Computes the absolute value of a complex number.

Syntax

```
rfx::Abs_c z
```

Argument	Description
z	A list containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A single value, the absolute value of a complex number.

Example

```
set z [list 4 3]
puts [rfx::Abs_c $z]
#-> 5.0
```

rfx::Abs_v

Computes the absolute values of a list of complex numbers, and also computes the absolute values in units of 10 dB or 20 dB.

Syntax

```
rfx::Abs_v -out <list_name> -z <array_name> [-dB 0 | 10 | 20]
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <list_name>`	Name of a list to store the list of absolute values. (List name, no default)
`-z <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the complex numbers. (Array name, no default)
`-dB 0 | 10 | 20`	Specifies the decibel level for absolute values. If `-dB` is not specified or `-dB 0` is specified, absolute values are computed on the linear scale. Default: `0`
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Z(Re) [list 1 1 0 -1 -1 -1 0 1]
set Z(Im) [list 0 1 1 1 0 -1 -1 -1]
rfx::Abs_v -out absvals -z Z
puts "abs values = $absvals"
#-> abs values = 1.0 1.414 1.0 1.414 1.0 1.414 1.0 1.414
```

```
set Z(Re) [list 0 0 0 0 0]
set Z(Im) [list 1e0 1e1 1e2 1e3 1e4]
rfx::Abs_v -out dBs -z Z -dB 10
puts "dBs= $dBs"
#-> 0.0 10.0 20.0 30.0 40.0
```

rfx::Abs2_c

Computes the square of the absolute value of a complex number.

Syntax

```
rfx::Abs2_c z
```

Argument	Description
z	A list containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A single value, the square of the absolute value of a complex number.

Example

```
set z [list 4 3]
puts [rfx::Abs2_c $z]
#-> 25
```

rfx::Abs2_v

Computes the square of the absolute value of a list of complex numbers.

Syntax

```
rfx::Abs2_v -out <list_name> -z <array_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <list_name>`	Name of a list to store the list of the square of absolute values. (List name, no default)
`-z <array_name>`	Name of an array containing list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the complex numbers. (Array name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Z(Re) [list 1 1 0 -1 -1 -1 0 1]
set Z(Im) [list 0 1 1 1 0 -1 -1 -1]
rfx::Abs2_v -out abs2 -z Z
puts "square of abs values = $abs2"
#-> square of abs values = 1.0 2.0 1.0 2.0 1.0 2.0 1.0 2.0
```

rfx::Add_c

Adds two complex numbers.

Syntax

```
rfx::Add_c z1 z2
```

Argument	Description
`z1 z2`	Two lists, each containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A list containing the real and imaginary parts of the sum (a complex number).

Example

```
set z1 [list 1 0]
set z2 [list 0 1]
puts [rfx::Add_c $z1 $z2]
#-> 1 1
```

rfx::Add_v

Adds two lists of complex numbers.

Syntax

```
rfx::Add_v -out <array_name> -z1 <array_name> -z2 <array_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the sum of the list of complex numbers specified using the keywords `-z1` and `-z2`. (Array name, no default)
`-z1 <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts of the complex numbers, respectively. (Array name, no default)
`-z2 <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts of the complex numbers, respectively. (Array name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Z1(Re) [list 0 1]
set Z1(Im) [list 1 2]
set Z2(Re) [list 1 2]
set Z2(Im) [list 3 4]
rfx::Add_v -out Z -z1 Z1 -z2 Z2
puts "Z(Re)= $Z(Re)"
puts "Z(Im)= $Z(Im)"
#-> Z(Re)= 1 3
#-> Z(Im)= 4 6
```

rfx::Cart2Polar_c

Converts a complex number from Cartesian to polar coordinates.

Syntax

```
rfx::Cart2Polar_c z
```

Argument	Description
z	A list containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A list containing the absolute value and the phase of the complex number.

Example

```
set z [list 1 1]
set polar [rfx::Cart2Polar_c $z]
puts "abs value = [format %.3f [lindex $polar 0]]"
puts "phase = [lindex $polar 1]"
#-> abs value = 1.414
#-> phase = 45.0
```

rfx::Cart2Polar_v

Converts a list of complex numbers from Cartesian to polar coordinates.

Syntax

```
rfx::Cart2Polar_v -out <array_name> -z <array_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements Abs and Phase. The values of the Abs element and the Phase element are the absolute value and the phase, respectively, of the list complex numbers specified using the keyword `-z`. (Array name, no default)
`-z <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the list of complex numbers. (Array name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Z(Re) [list 1 1 0 -1 -1 -1 0 1]
set Z(Im) [list 0 1 1 1 0 -1 -1 -1]
rfx::Cart2Polar_v -out polar -z Z
puts "abs values = $polar(Abs)"
puts "phases = $polar(Phase)"
#-> abs values = 1.0 1.414 1.0 1.414 1.0 1.414 1.0 1.414
#-> phases = 0.0 45.0 89.999 135.0 180.0 -135.0 -89.999 -45.0
```

rfx::Conj_c

Computes the complex conjugate of a complex number.

Syntax

```
rfx::Conj_c z
```

Argument	Description
z	A list containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A list containing the real and imaginary parts of the complex conjugate.

Example

```
set z [list 1 1]
puts [rfx::Conj_c $z]
#-> 1 -1
```

rfx::Conj_v

Computes the complex conjugate of a list of complex numbers.

Syntax

```
rfx::Conj_v -out <array_name> -z <array_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the complex conjugate of the list of complex numbers specified using the keyword `-z`. (Array name, no default)
`-z <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the complex numbers. (Array name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Z(Re) [list 1 1]
set Z(Im) [list -1 1]
rfx::Conj_v -out Conj -z Z
puts "Real part of complex conjugate= $Conj(Re)"
puts "Imaginary part of complex conjugate= $Conj(Im)"
#-> Real part of complex conjugate= 1 1
#-> Imaginary part of complex conjugate= 1 -1
```

rfx::Div_c

Divides two complex numbers.

Syntax

```
rfx::Div_c z1 z2
```

Argument	Description
`z1 z2`	Two lists, each containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A list containing the real and imaginary parts of the quotient (a complex number).

Example

```
set z1 [list 4 0]
set z2 [list 0 2]
puts [rfx::Div_c $z1 $z2]
#-> 0.0 -2.0
```

rfx::Div_v

Divides two lists of complex numbers.

Syntax

```
rfx::Div_v -out <array_name> -z1 <array_name> -z2 <array_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the quotient of the list of complex numbers specified using the keywords `-z1` and `-z2`. (Array name, no default)
`-z1 <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts of the complex numbers, respectively. (Array name, no default)
`-z2 <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts of the complex numbers, respectively. (Array name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Z1(Re) [list 4 4]
set Z1(Im) [list 7 2]
set Z2(Re) [list 1 3]
set Z2(Im) [list -3 -1]
rfx::Div_v -out Z -z1 Z1 -z2 Z2
puts "Z(Re)= $Z(Re)"
puts "Z(Im)= $Z(Im)"
#-> Z(Re)= -1.7 1.0
#-> Z(Im)= 1.9 1.0
```

rfx::Im_c

Computes the imaginary part of a complex number.

Syntax

```
rfx::Im_c z
```

Argument	Description
z	A list containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A single value, the imaginary part of a complex number.

Example

```
set z [list 1 2]
puts [rfx::Im_c $z]
#-> 2
```

rfx::Mul_c

Multiplies two complex numbers.

Syntax

```
rfx::Mul_c z1 z2
```

Argument	Description
`z1 z2`	Two lists, each containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A list containing the real and imaginary parts of the product (a complex number).

Example

```
set z1 [list 1 0]
set z2 [list 0 1]
puts [rfx::Mul_c $z1 $z2]
#-> 0 1
```

rfx::Mul_v

Multiplies two lists of complex numbers.

Syntax

```
rfx::Mul_v -out <array_name> -z1 <array_name> -z2 <array_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the product of the list of complex numbers specified using the keywords `-z1` and `-z2`. (Array name, no default)
`-z1 <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts of the complex numbers, respectively. (Array name, no default)
`-z2 <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts of the complex numbers, respectively. (Array name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Z1(Re) [list 4 4]
set Z1(Im) [list 7 2]
set Z2(Re) [list 1 3]
set Z2(Im) [list -3 -1]
rfx::Mul_v -out Z -z1 Z1 -z2 Z2
puts "Z(Re)= $Z(Re)"
puts "Z(Im)= $Z(Im)"
#-> Z(Re)= 25 14
#-> Z(Im)= -5 2
```

rfx::Mulsc_c

Multiplies a scalar and a complex number.

Syntax

```
rfx::Mulsc_c c z
```

Argument	Description
`c`	A real number. (Real number, no default)
`z`	A list containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A list containing the real and imaginary parts of the product (a complex number) of the scalar and the complex number.

Example

```
set c 5.0
set z [list 2.0 3.0]
puts [rfx::Mulsc_c $c $z]
#-> 10.0 15.0
```

rfx::Phase_c

Computes the principal value of the phase (in degrees, in the interval $(-180°, 180°]$) of a complex number specified in Cartesian coordinates.

Syntax

```
rfx::Phase_c z
```

Argument	Description
z	A list containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A single value, the phase of a complex number.

Example

```
set z [list 1 1]
puts [rfx::Phase_c $z]
#-> 45.0
```

rfx::Phase_v

Computes the principal value of the phase (in degrees, in the interval $(-180°, 180°]$) of a list of complex numbers specified in Cartesian coordinates.

Syntax

```
rfx::Phase_v -out <list_name> -z <array_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <list_name>`	Name of a list to store the list of phases. (List name, no default)
`-z <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the complex numbers. (Array name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Z(Re) [list 1 1 0 -1 -1 -1 0 1]
set Z(Im) [list 0 1 1 1 0 -1 -1 -1]
rfx::Phase_v -out phases -z Z
puts "phases = $phases"
#-> phases = 0.0 45.0 89.999 135.0 180.0 -135.0 -89.999 -45.0
```

rfx::Polar2Cart_c

Converts a complex number from polar to Cartesian coordinates.

Syntax

```
rfx::Polar2Cart_c z
```

Argument	Description
z	A list containing the absolute value and the phase of a complex number. (List of real numbers, no default)

Returns

A list containing the real and imaginary parts of a complex number.

Example

```
set z [list 1.414 45]
set ReIm [rfx::Polar2Cart_c $z]
puts "Real part = [lindex $ReIm 0]"
puts "Imaginary part = [lindex $ReIm 1]"
#-> Real part = 0.999
#-> Imaginary part = 0.999
```

rfx::Polar2Cart_v

Converts a list of complex numbers from polar to Cartesian coordinates.

Syntax

```
rfx::Polar2Cart_v -out <array_name> -z <array_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the list of complex numbers specified using the keyword `-z`. (Array name, no default)
`-z <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Abs and Phase. The values of the Abs element and the Phase element are the absolute value and the phase (in degrees), respectively, of the list of complex numbers. (Array name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Z(Abs) [list 1.414 1]
set Z(Phase) [list 45 60]
rfx::Polar2Cart_v -out ReIm -z Z
puts "Real part = $ReIm(Re)"
puts "Imaginary part = $ReIm(Im)"
```

```
#-> Real part = 0.999 0.5
#-> Imaginary part = 0.999 0.866
```

rfx::Re_c

Computes the real part of a complex number.

Syntax

```
rfx::Re_c z
```

Argument	Description
z	A list containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A single value, the real part of a complex number.

Example

```
set z [list 1 2]
puts [rfx::Re_c $z]
#-> 1
```

rfx::Sign

Computes the sign of a real number.

Syntax

```
rfx::Sign r1
```

Argument	Description
r1	A real value. (Real number, no default)

Returns

A single value, the sign of a real number.

Example

```
puts [rfx::Sign -2]
#-> -1.0
```

rfx::Sub_c

Subtracts two complex numbers.

Syntax

```
rfx::Sub_c z1 z2
```

Argument	Description
`z1 z2`	Two lists, each containing the real and imaginary parts of a complex number. (List of real numbers, no default)

Returns

A list containing the real and imaginary parts of the difference (a complex number).

Example

```
set z1 [list 1 0]
set z2 [list 0 1]
puts [rfx::Sub_c $z1 $z2]
#-> 1 -1
```

rfx::Sub_v

Subtracts two lists of complex numbers.

Syntax

```
rfx::Sub_v -out <array_name> -z1 <array_name> -z2 <array_name>
   [-info 0 | 1 | 2 | 3] [-help 0 | 1]
```

Argument	Description
`-out <array_name>`	Name of an array to store the results. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts, respectively, of the difference of the list of complex numbers specified using the keywords `-z1` and `-z2`. (Array name, no default)
`-z1 <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts of the complex numbers, respectively. (Array name, no default)
`-z2 <array_name>`	Name of an array containing a list of complex numbers. The array has one string-valued index. The index contains the elements Re and Im. The values of the Re element and the Im element are the real and imaginary parts of the complex numbers, respectively. (Array name, no default)
`-info 0 | 1 | 2 | 3`	Sets local info level. Default: `0`
`-help 0 | 1`	Prints a help screen if set to `1`. Default: `0`

Returns

None.

Example

```
set Z1(Re) [list 1 1]
set Z1(Im) [list 0 2]
set Z2(Re) [list 0 2]
set Z2(Im) [list 1 4]
rfx::Sub_v -out Z -z1 Z1 -z2 Z2
puts "Z(Re)= $Z(Re)"
puts "Z(Im)= $Z(Im)"
#-> Z(Re)= 1 -1
#-> Z(Im)= -1 -2
```

lib::SetInfoDef

Sets the default information level.

NOTE Level 0: Warning, error, or status messages only.
Level 1: Echo results.
Level 2: Show progress and some debug information.
Level 3: Show all debug information.

The local info level also can be set using the `-info` keyword of the procedures in the RF extraction library.

Syntax

```
lib::SetInfoDef 0 | 1 | 2 | 3
```

Argument	Description
`<info_level>`	Sets the default info level. Default: `0`

Returns

None.

Example

```
lib::SetInfoDef 2
```

References

[1] H. Hillbrand and P. H. Russer, “An Efficient Method for Computer Aided Noise Analysis of Linear Amplifier Networks,” *IEEE Transactions on Circuits and Systems*, vol. CAS-23. no. 4, pp. 235–238, 1976.

[2] M. Reisch, *High-Frequency Bipolar Transistors: Physics, Modeling, Applications*, Berlin: Springer, 2003.

[3] B. Razavi, *Design of Analog CMOS Integrated Circuits*, Boston: McGraw-Hill, 2001.

[4] *Sentaurus™ Device User Guide*, Version N-2017.09, Mountain View, California: Synopsys, Inc., 2017.

[5] R. S. Carson, *High-Frequency Amplifiers*, New York: John Wiley & Sons, 2nd ed., 1982.

[6] R. Ludwig and G. Bogdanov, *RF Circuit Design: Theory and Applications*, Upper Saddle River, New Jersey: Prentice Hall, 2nd ed., 2009.

[7] W. Liu, *Handbook of III-V Heterojunction Bipolar Transistors*, New York: John Wiley & Sons, 1998.

[8] M. S. Gupta, "Power Gain in Feedback Amplifiers, a Classic Revisited," *IEEE Transactions on Microwave Theory and Techniques*, vol. 40. no. 5, pp. 864–879, 1992.

[9] J. D. Cressler and G. Niu, *Silicon-Germanium Heterojunction Bipolar Transistors*, Boston: Artech House, 2003.

[10] A. M. Niknejad, *Electromagnetics for High-Speed Analog and Digital Communication Circuits*, Cambridge: Cambridge University Press, 2007.

[11] G. Gonzalez, *Microwave Transistor Amplifiers: Analysis and Design*, Upper Saddle River, New Jersey: Prentice Hall, 2nd ed., 1997.

[12] V. Rizzoli and A. Lipparini, "Computer-Aided Noise Analysis of Linear Multiport Networks of Arbitrary Topology," *IEEE Transactions on Microwave Theory and Techniques*, vol. MTT-33. no. 12, pp. 1507–1512, 1985.

[13] M. E. Mokari and W. Patience, "A New Method of Noise Parameter Calculation Using Direct Matrix Analysis," *IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications*, vol. 39. no. 9, pp. 767–771, 1992.

[14] F. Bonani and G. Ghione, *Noise in Semiconductor Devices, Modeling and Simulation*, Berlin: Springer, 2001.

APPENDIX I PhysicalConstants Library

This appendix provides information about the PhysicalConstants library.

Major Physical Constants

This library defines a set of variables of major physical constants [1] (see Table 36).

Table 36 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
RydbergConstant	1.097373153413e7	m^{-1}

Table 36 Variables defined in PhysicalConstants library

Name of variable	Value	Unit
`SpeedOfLight`	299792458	m/s
`StefanBoltzmannConstant`	5.6705119e-8	$W/m^2/K^4$
`kT300`	0.0258521592446	V

To load the library, use the command:

```
load_library physicalconstants
```

The PhysicalConstants library uses a unique namespace identifier (`const::`) for its variables. All variables associated with this library are accessed with the namespace identifier prepended:

```
const::<var_name>
```

For example:

```
load_library physicalconstants
puts "c=$const::SpeedOfLight"
#-> c=299792458
```

The following conventions are used for the syntax of Tcl commands:

- Angle brackets – <> – indicate text that must be replaced, but they are *not* part of the syntax.

References

[1] G. Woan, *The Cambridge Handbook of Physics Formulas*, Cambridge: Cambridge University Press, 2000.