

© Agilent Technologies, Inc. 2000-2011

5301 Stevens Creek Blvd., Santa Clara, CA 95052 USA

No part of this documentation may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Agilent Technologies, Inc. as governed by United States and international copyright laws.

Acknowledgments

Mentor Graphics is a trademark of Mentor Graphics Corporation in the U.S. and other countries. Mentor products and processes are registered trademarks of Mentor Graphics Corporation. * Calibre is a trademark of Mentor Graphics Corporation in the US and other countries. "Microsoft®, Windows®, MS Windows®, Windows NT®, Windows 2000® and Windows Internet Explorer® are U.S. registered trademarks of Microsoft Corporation. Pentium® is a U.S. registered trademark of Intel Corporation. PostScript® and Acrobat® are trademarks of Adobe Systems Incorporated. UNIX® is a registered trademark of the Open Group. Oracle and Java and registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. SystemC® is a registered trademark of Open SystemC Initiative, Inc. in the United States and other countries and is used with permission. MATLAB® is a U.S. registered trademark of The Math Works, Inc.. HiSIM2 source code, and all copyrights, trade secrets or other intellectual property rights in and to the source code in its entirety, is owned by Hiroshima University and STARC. FLEXIm is a trademark of Globetrotter Software, Incorporated. Layout Boolean Engine by Klaas Holwerda, v1.7 <http://www.xs4all.nl/~kholwerd/bool.html> . FreeType Project, Copyright (c) 1996-1999 by David Turner, Robert Wilhelm, and Werner Lemberg. QuestAgent search engine (c) 2000-2002, JObjects. Motif is a trademark of the Open Software Foundation. Netscape is a trademark of Netscape Communications Corporation. Netscape Portable Runtime (NSPR), Copyright (c) 1998-2003 The Mozilla Organization. A copy of the Mozilla Public License is at <http://www.mozilla.org/MPL/> . FFTW, The Fastest Fourier Transform in the West, Copyright (c) 1997-1999 Massachusetts Institute of Technology. All rights reserved.

The following third-party libraries are used by the NlogN Momentum solver:

"This program includes Metis 4.0, Copyright © 1998, Regents of the University of Minnesota", <http://www.cs.umn.edu/~metis> , METIS was written by George Karypis (karypis@cs.umn.edu).

Intel® Math Kernel Library, <http://www.intel.com/software/products/mkl>

SuperLU_MT version 2.0 - Copyright © 2003, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from U.S. Dept. of Energy). All rights reserved. SuperLU Disclaimer: 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 COPYRIGHT OWNER 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.

7-zip - 7-Zip Copyright: Copyright (C) 1999-2009 Igor Pavlov. Licenses for files are: 7z.dll: GNU LGPL + unRAR restriction, All other files: GNU LGPL. 7-zip License: This library is free software; you can redistribute it and/or modify it under the terms of the GNU

Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA. unRAR copyright: The decompression engine for RAR archives was developed using source code of unRAR program. All copyrights to original unRAR code are owned by Alexander Roshal. unRAR License: The unRAR sources cannot be used to re-create the RAR compression algorithm, which is proprietary. Distribution of modified unRAR sources in separate form or as a part of other software is permitted, provided that it is clearly stated in the documentation and source comments that the code may not be used to develop a RAR (WinRAR) compatible archiver. 7-zip Availability: <http://www.7-zip.org/>

AMD Version 2.2 - AMD Notice: The AMD code was modified. Used by permission. AMD copyright: AMD Version 2.2, Copyright © 2007 by Timothy A. Davis, Patrick R. Amestoy, and Iain S. Duff. All Rights Reserved. AMD License: Your use or distribution of AMD or any modified version of AMD implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. AMD Availability: <http://www.cise.ufl.edu/research/sparse/amd>

UMFPACK 5.0.2 - UMFPACK Notice: The UMFPACK code was modified. Used by permission. UMFPACK Copyright: UMFPACK Copyright © 1995-2006 by Timothy A. Davis. All Rights Reserved. UMFPACK License: Your use or distribution of UMFPACK or any modified version of UMFPACK implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. UMFPACK Availability: <http://www.cise.ufl.edu/research/sparse/umfpack> UMFPACK (including versions 2.2.1 and earlier, in FORTRAN) is available at <http://www.cise.ufl.edu/research/sparse>. MA38 is available in the Harwell Subroutine Library. This version of UMFPACK includes a modified form of COLAMD Version 2.0, originally released on Jan. 31, 2000, also available at

<http://www.cise.ufl.edu/research/sparse> . COLAMD V2.0 is also incorporated as a built-in function in MATLAB version 6.1, by The MathWorks, Inc. <http://www.mathworks.com> . COLAMD V1.0 appears as a column-preordering in SuperLU (SuperLU is available at <http://www.netlib.org>). UMFPACK v4.0 is a built-in routine in MATLAB 6.5. UMFPACK v4.3 is a built-in routine in MATLAB 7.1.

Qt Version 4.6.3 - Qt Notice: The Qt code was modified. Used by permission. Qt copyright: Qt Version 4.6.3, Copyright (c) 2010 by Nokia Corporation. All Rights Reserved. Qt License: Your use or distribution of Qt or any modified version of Qt implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the

terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User

documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission."

Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. Qt Availability: <http://www.qtsoftware.com/downloads> Patches Applied to Qt can be found in the installation at:

\$HPEESOF_DIR/prod/licenses/thirdparty/qt/patches. You may also contact Brian Buchanan at Agilent Inc. at brian_buchanan@agilent.com for more information.

The HiSIM_HV source code, and all copyrights, trade secrets or other intellectual property rights in and to the source code, is owned by Hiroshima University and/or STARC.

Errata The ADS product may contain references to "HP" or "HPEESOF" such as in file names and directory names. The business entity formerly known as "HP EEsof" is now part of Agilent Technologies and is known as "Agilent EEsof". To avoid broken functionality and to maintain backward compatibility for our customers, we did not change all the names and labels that contain "HP" or "HPEESOF" references.

Warranty The material contained in this document is provided "as is", and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this documentation and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license. Portions of this product include the SystemC software licensed under Open Source terms, which are available for download at <http://systemc.org/> . This software is redistributed by Agilent. The Contributors of the SystemC software provide this software "as is" and offer no warranty of any kind, express or implied, including without limitation warranties or conditions or title and non-infringement, and implied warranties or conditions merchantability and fitness for a particular purpose. Contributors shall not be liable for any damages of any kind including without limitation direct, indirect, special,

incidental and consequential damages, such as lost profits. Any provisions that differ from this disclaimer are offered by Agilent only.

Restricted Rights Legend U.S. Government Restricted Rights. Software and technical data rights granted to the federal government include only those rights customarily provided to end user customers. Agilent provides this customary commercial license in Software and technical data pursuant to FAR 12.211 (Technical Data) and 12.212 (Computer Software) and, for the Department of Defense, DFARS 252.227-7015 (Technical Data - Commercial Items) and DFARS 227.7202-3 (Rights in Commercial Computer Software or Computer Software Documentation).

DRC Quick Start	8
DRC Message Window	8
DRC Engine	9
Rule Registry File	9
Rule Directories	10
Setting Up a Quick DRC	10
Setting Up a DRC with Predefined Rules	11
Viewing DRC Results	12
Opening a DRC Example Workspace	14
Setting DRC Memory Use and Performance	16
Writing Design Rules	19
Extension and Intrusion Definitions	19
Anatomy of a Simple DRC Rule File	19
Layer Management	20
Recognizing Devices in Flattened Layouts	23
Complete DRC Example	23
Additional DRC Examples	32
DRC Functions (alphabetical)	34
DRC Functions (by category)	37
Import and Export Layers	37
Edge Selection	37
Edge Operations	38
Polygon Selection	38
Polygon Operations	39
DRC Job Management	39
DRC Job Management	40
dedrc_run_drc_ex	40
DRC Match Functions for Error Checking	41
Match AEL Function	41
Complete Match AEL Function	41
Running Match AEL function	42
Boolean Operations on Edges	43
all_edges()	43
invert_edges()	43
Conditional Selection	46
compensate()	46
contains()	49
corner_edges()	51
Edge Selection Based on Grid	53
de_touch()	53
double_clearance()	54
dve_combine()	56
Edge Selection Based On Clearance	57
dve_drc()	58
dve_drc_group()	58
dve_segsize()	59
Edge Qualifiers	61
Edge Selection Based on Corners	69
external()	69
gap()	71
internal()	73
nests()	74
notch()	77
off_grid()	79
Edge Compensation	80
single_clearance()	80
spacing()	83
width()	86
DRC Layer Management Commands	89

dve_import_text_layer()	89
dve_export_layer()	90
dve_identify_cell_layer()	90
dve_import_cell_layer()	91
dve_import_layer()	92
Macros	93
intrusion()	93
protrusion()	94
Merge Operations on Polygons	96
dve_bool_and()	96
dve_bool_not()	97
dve_bool_or()	97
dve_merge()	98
dve_self()	99
dve_self_merge()	99
Example for Performing Boolean Operations	100
Operations for Polygon Extraction from Edges	101
dve_plgout()	101
dve_quadout()	101
Polygon Extraction	107
dve_plg_extract()	107
Polygon Selection	108
Polygon Selection Based on Intrinsic Properties	108
poly_area()	108
poly_edge_code()	109
poly_hole_count()	111
poly_inter_layer()	114
poly_line_length()	116
poly_path_count()	117
poly_path_length()	118
Polygon Selection Based on Edge Relationships	120
poly_perimeter()	121
Polygon Selection Based on Merge Properties	121
Sizing Operations on Polygons	124
dve_oversize()	124
dve_undersize ()	125
Troubleshooting Design Rule Checker	126
Layer Management Errors (101-199)	126
Layer Management Warnings (201-299)	127
Command Usage Errors (301-399)	127
Command Usage Warnings (401-499)	129
Using Calibre DRC Link	131
Running Calibre DRC in Local Mode	131
Running Calibre DRC in Remote Mode	132
Viewing Calibre DRC Link Results	134
Using Assura DRC Link	136
Running Assura DRC in Local Mode	136
Running Assura DRC in Remote Mode	137
Viewing Assura DRC Link Results	139

DRC Quick Start

The Design Rule Checker helps ensure that a layout design conforms to the physical constraints required to produce it. These constraints can be a requirement of the design itself (such as reducing noise) or a requirement of the process used to produce the design.

You can run a quick check to ensure conformance to basic design requirements, such as minimum width and spacing, or you can run a check using custom rules to ensure that a design meets manufacturing specifications. In either case, you can check all or part of the design.

You can run a design rule check using links to external DRC engines. For more information on using external DRC links, see *Using Calibre DRC Link (drc)* and *Using Assura DRC Link (drc)*.

Whether you run a quick check or a check using custom rules, the procedure is essentially the same:

- Define or select a design rule.
- Run the design check using the defined or selected rule.
- Load the results.
- View any errors that were found.

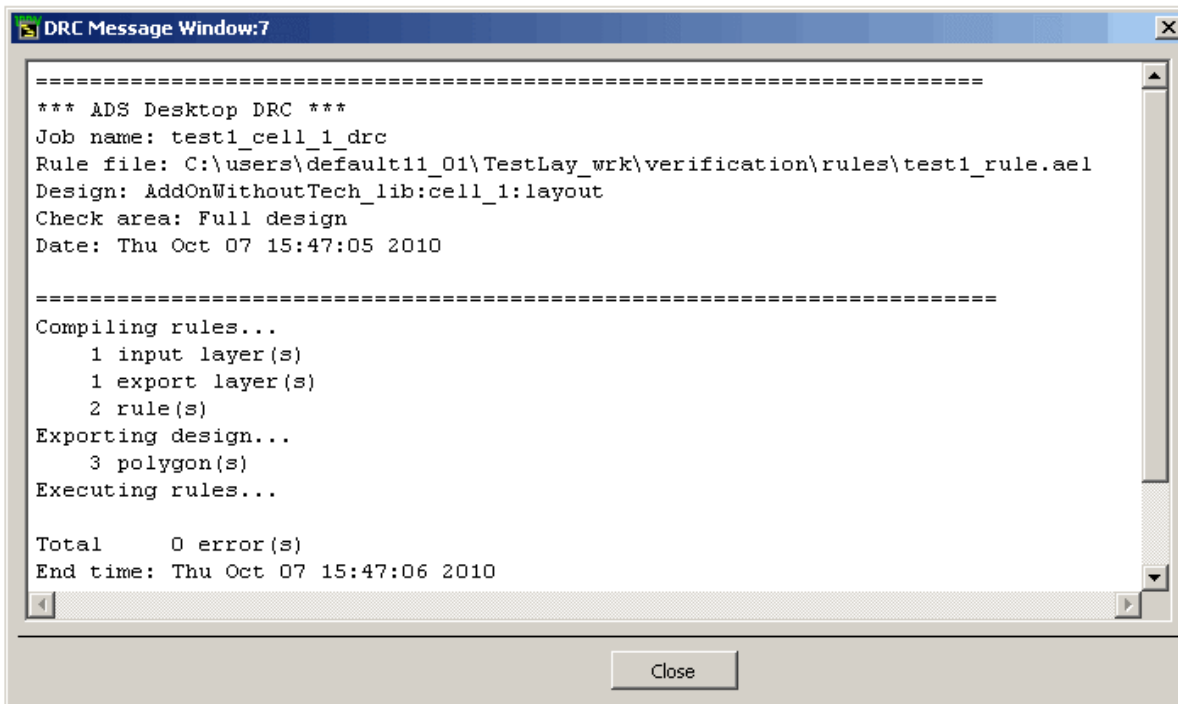


Note

Layout resolution must be set properly for DRC to work on designs. The layout is drawn at the resolution specified in the **Options > Preferences** dialog box. DRC works at this resolution and cannot find clearance violations below the resolution value. To have DRC check clearance rules at lower values, change the layout resolution to a value lower than the smallest DRC rule.

DRC Message Window

The DRC Message window provides information on the status of the current Design Rule check. The window displays as you set up and run a DRC and then displays a summary of in the View Errors panel of the DRC or Custom DRC dialog box.



DRC Engine

In ADS, you may run a design rule check using the ADS DRC engine, or you may link to either **Calibre** or **Assura** via the **DRC Engine** drop-down list. For more information on using external DRC links, see *Using Calibre DRC Link (drc)* and *Using Assura DRC Link (drc)*.

Rule Registry File

A rule registry file, called *setrule.ael*, is required in a rule directory to display the list of available rule files by file names.

The format of *setrule.ael* is as follows:

```

dve_set_rule_list(list(
<rule_name>,<rule_file>,
<rule_name>,<rule_file>,
.....
,
));

```

where *<rule_name>* can be a string that briefly describes the purpose of the rules and *<rule_file>* is the actual file name.

For example, if you create a *setrule.ael* file for the Workspace directory, you can select the Workspace **Rule location** to display this list of rules.

```

// Rule Registry File
dve_set_rule_list(list(
"Substrate Via Design Rule", "viaRule.ael",
"NiCr Thin Film Resistor Design Rule", "resistorRule.ael",
"Gate Metal Spacing Rule", "gateSpacing.ael"

```

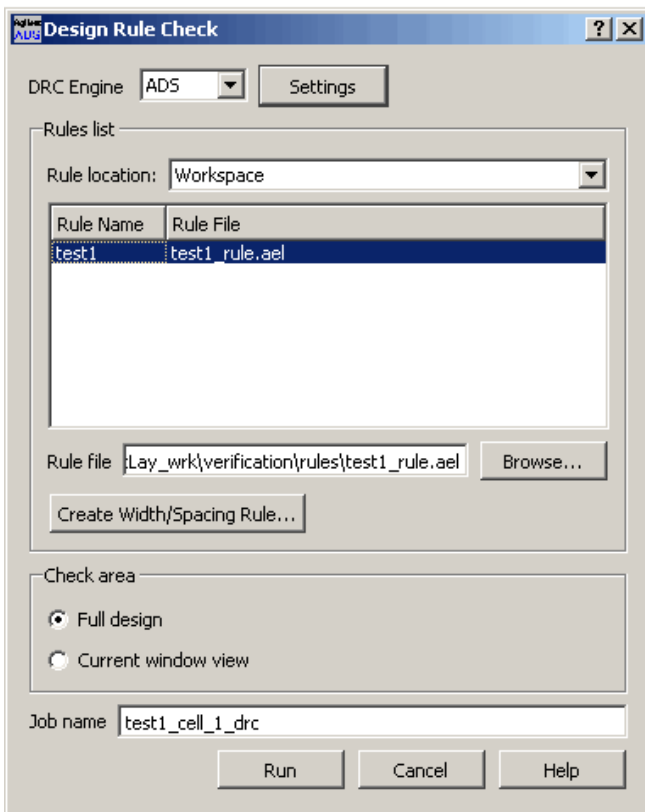
));

For Calibre DRC use *dve_set_calibre_rule_list()* and for Assura DRC use *dve_set_assura_rule_list()* in setrule.ael file.

Rule Directories

You can store a rule file in any directory and find the file using the **Rule file** browser. You can click the Rule file **Browse** button to find a rule file in any directory. However, it is better to use one of the four rule directories supported by the program to facilitate rule file browsing.

The pre-defined rule directories for storing custom rules are: Site, User, Workspace and any installed design kit. The **Rule Location** drop-down menu corresponding to these directories is available in the dialog box. Design rules can be placed at any of these locations.

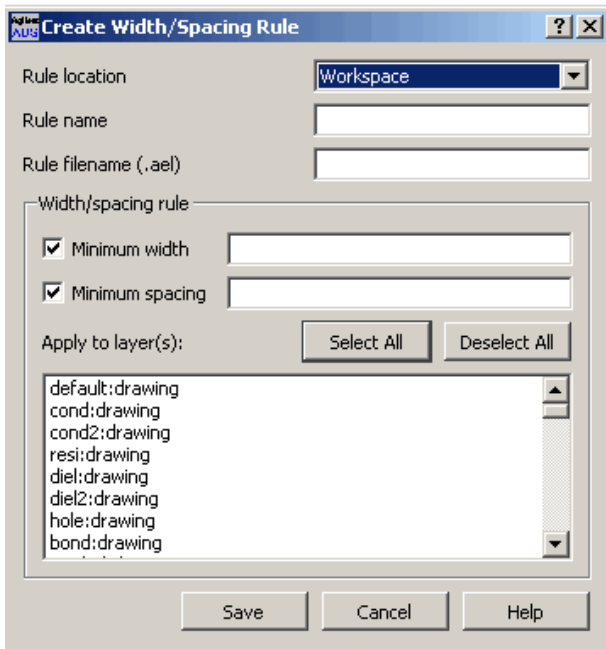


Setting Up a Quick DRC

You can use a quick DRC to check selected components or to check an entire design against basic design requirements. After you provide the information needed, the program writes a design rule for you that you can save and reuse again.

To set up a quick DRC:

1. In the Layout window, create a new layout or open an existing one.
2. From the menu, choose **Tools > DRC** to open the Design Rule Check dialog box.
3. Click the **Create Width/Spacing Rule** button to define a basic design rule to be used. The Create Width/Spacing Rule dialog box appears.



4. Select the **Rule location** from the drop-down list and then enter an appropriate Rule name and Rule filename.
5. Select the parameter(s) you want to check and enter a value in the selected field. Do not include units when you enter a value in this panel.
 - *Minimum Width* defines the narrowest allowable value in the design.
 - *Minimum Spacing* defines the narrowest allowable spacing between shapes in the design.

**Note**

Design Rule Checker will run most efficiently if reasonable values are set for Minimum Width and Minimum Spacing. Values that are much larger than the actual design will create longer processing times.

6. In the **Apply to Layer(s)** list, select the design layer(s) you want checked.
 - *Apply to Layer(s)* displays a list of the layers in the current design. Choose the layers that you want the rule to apply to.
7. Click **Save** to save the Width/Spacing Rules and dismiss the Create Width/Spacing Rule dialog box.

When you save a Design Rule, the program automatically updates the rule registry file to include the new rule (see [Rule Registry File](#)).
8. Click **Run** in the Design Rule Check dialog box to start the process.

Setting Up a DRC with Predefined Rules

Typically you use a DRC with predefined rules to check a design against a manufacturing specification. A DRC with predefined rules differs from a quick DRC in two major ways:

- You specify a custom design rule.
- You must create a DRC layer in the design on which to display error segments.

To set up a DRC with predefined rules:

1. In the **Layout** window, open an existing layout or create a new one.
2. Choose **Options > Layers**. If the design does not have a *drc* layer, create one. Refer to *Layer Management (drc)* for more information.
3. Choose **Tools > DRC** to open the Design Rule Check dialog box.
4. Define the Rules using the *Rules list* section of the dialog.

- The **Rule location** enables you to select a DRC rules file from one of several predefined locations: Site, User, Workspace, or a design kit.
 - The **Rule Name** list box includes a list of available rule files and their associated AEL file name. You can select one of these files by simply clicking the file name.
 - The **Rule file** field displays the selected rules file.
 - The **Browse** button enables you to display the Open DRC Rules File dialog box where you can select a rules file from other locations, open the file, and view/edit the contents.
 - The **Create Width/Spacing Rule** button launches the Create Width/Spacing Rule dialog box which enables you to define a new rule. For details, see [Setting Up a Quick DRC](#).
5. Choose Site, User, Workspace, or a design kit to view predefined rules for that location (see [Rule Registry File](#)). If more than one design kit is enabled, each design kit will be listed in the **Rule location** list box.
 6. Select a **Rule Name** from the Rules List. If the rule file you want is not in the rule registry for a pre-defined location, you can browse to find it or you can enter the path and rule file name in the **Rule File** field.
 7. Select the **Check area** for Full design or Current window view.
 - **Full design** - Check the entire layout.
 - **Current window view** - Check only the area that is currently visible in the Layout window.

 **Hint**
You can save time on large designs by checking only the area of concern.

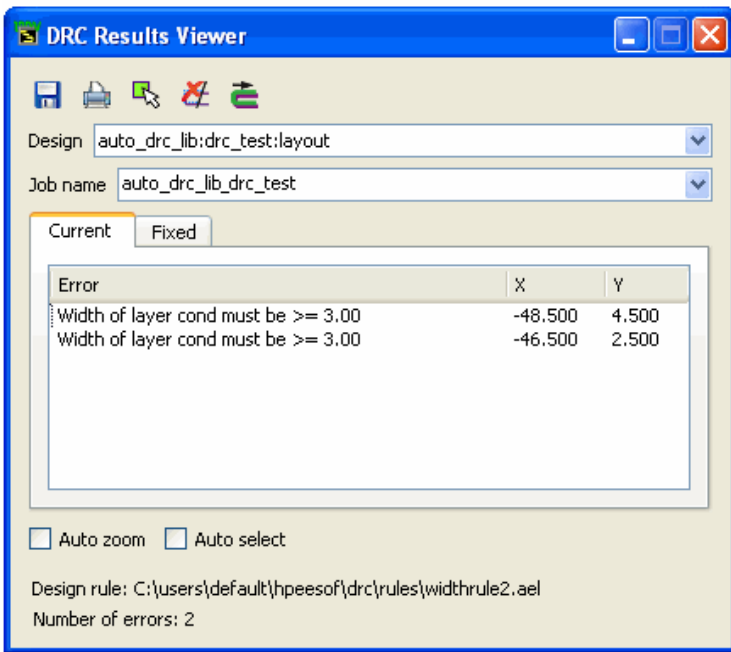
8. You can accept the default Job name or enter a different name. The default Job name is the design name with the suffix *_drc*.
9. Click **Run** to compile the selected rule and start the process. The DRC Message window appears.

A message similar to the example displays in the message window:

```
Run DRC Job _drc for full design...  
DRC process complete
```

Viewing DRC Results

After a DRC job completes successfully, the DRC Results Viewer dialog box will automatically appear.



If there are any DRC errors, each of the errors will be listed with an associated error ID number. You can select an error from the list to display the error in the Layout window. If there are no errors found, the list will be empty and the error count will be zero.

Note

For a quick DRC, the program automatically creates a drc layer on which error segments are displayed. For a DRC with predefined rules, the program does not create a drc layer automatically. You must create an appropriate drc layer(s) before you run the check. For more information refer to *Layer Management* (drc).

Because the DRC job is automatically saved to disk, you can review the DRC results at a later time.

To access and view results (if it has not already been opened by a DRC job):

1. Choose **Tools > DRC Results Viewer**. The results viewer is displayed.
2. In the **Design** drop-down list, select the design you want to view. Only designs in open windows are shown in the list.
3. In the **Job name** drop-down list, select the job name you want to view.
4. Select an error to view from the list. The error will be highlighted in the layout window.
5. Enable **Auto Select** and/or **Auto Zoom** as desired.
 - **Auto Zoom** - Shifts the design in the layout window to center the error.
 - **Auto Select** - Selects the error in the layout window. The error segments are selected so you can delete the DRC segments as you fix problems in the layout.
6. The Design rule file and the error count are all displayed at the bottom of the DRC Results Viewer dialog box.

To view specific error types:

1. In the Layout window, choose **Options > Preferences > Select**.
2. Turn off all Select Filters except the specific error type you want to view (for example, Polylines).
3. Click **Select by Cursor** and experiment with selecting errors by dragging a select box around areas in the layout where errors are indicated.
4. Click **OK** to dismiss the message window, **Cancel** to dismiss the dialog box.

To sort errors in the DRC Results Viewer:

Click on a field (i.e. ID, Error, X, or Y) in the header bar of the list. By clicking on one of these fields, the errors will be sorted based on the field clicked. For example, clicking the X will sort the errors based on the X location.

ID	Error	X	Y
18	Metal 0 Inclusion in Gate Metal min is 1 um	-19.000	200.000
19	Metal 0 Inclusion in Gate Metal min is 1 um	-19.000	120.000
44	Metal 0 Inclusion in Gate Metal min is 1 um	-19.000	-120.000
46	Metal 0 Inclusion in Gate Metal min is 1 um	-19.000	-200.000
20	Metal 0 Inclusion in Gate Metal min is 1 um	80.000	101.000
21	Metal 0 Inclusion in Gate Metal min is 1 um	80.000	219.000

To reverse the sort order, click the field again.

Note

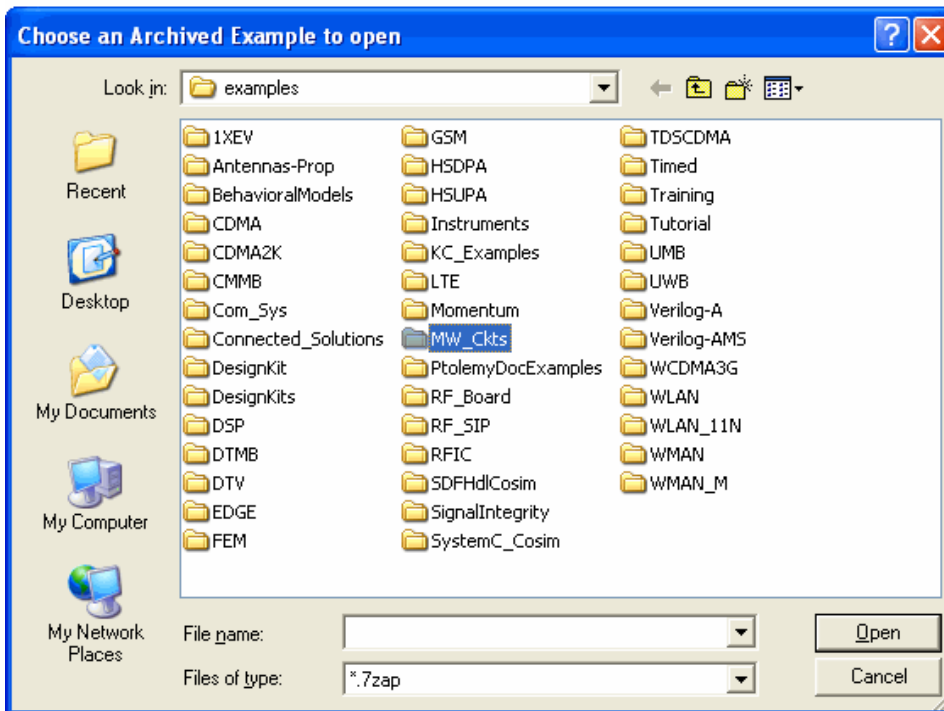
The X,Y location in the error list is just one of the points included in the error. This X,Y location is helpful in locating an error in a large design.

Opening a DRC Example Workspace

The Advanced Design System examples directory contains many DRC examples. Examples are constantly improved and new ones are added, so the files in your program may differ from what is shown here.

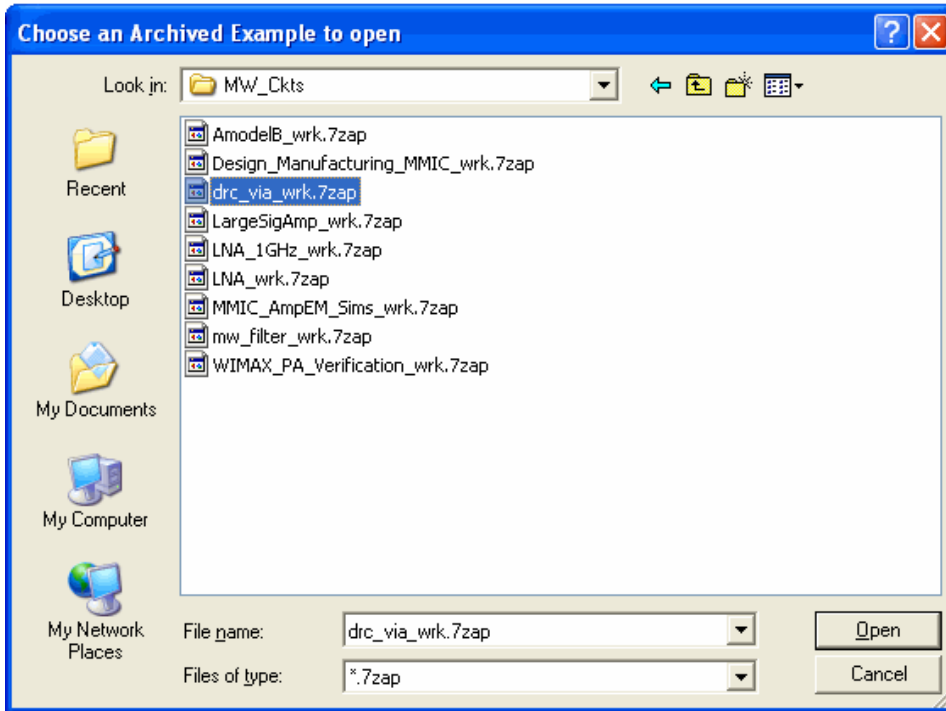
To open a DRC example workspace:

1. In the Main window, choose **File > Open > Example...**
2. In the DRC Example dialog box, select **MW_Ckts** folder.



3. Click **Open**.
4. In the Choose an Archived Example to Open dialog box, select the `drc_via_wrk.7zap`

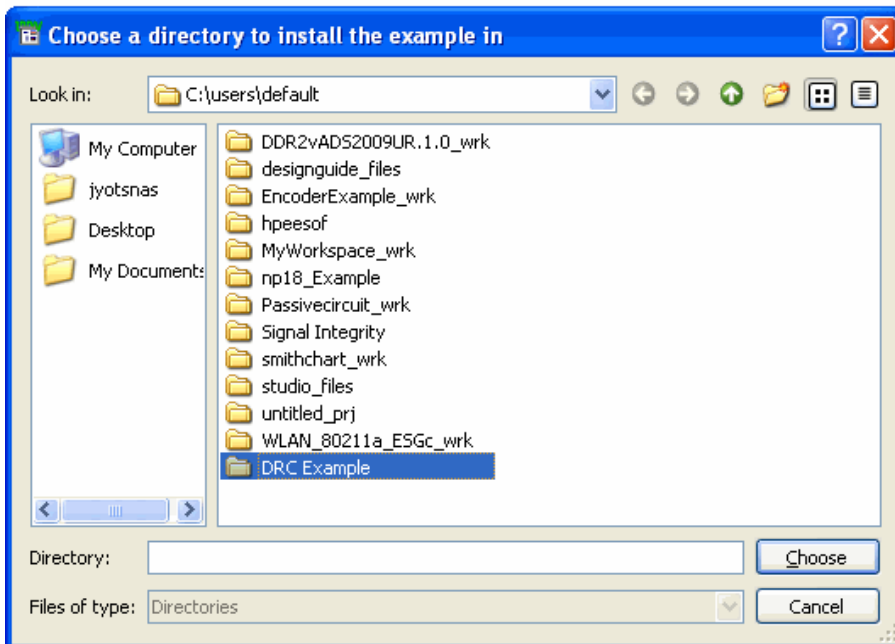
archived example.



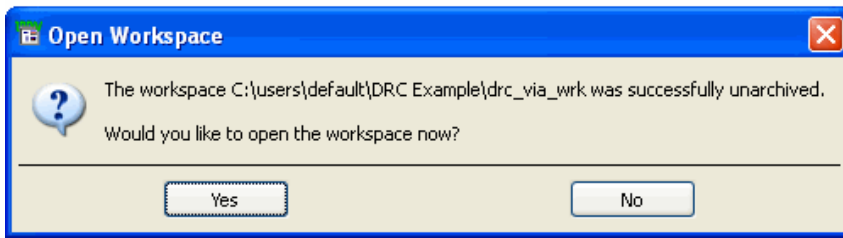
5. Click **Open**.
6. In the Choose a directory to install the example in dialog box, select the directory (for example, DRC Example) where the DRC example must be unarchived.

Note

If you are running ADS on network, you must have the write permission to install or unarchive the workspace.



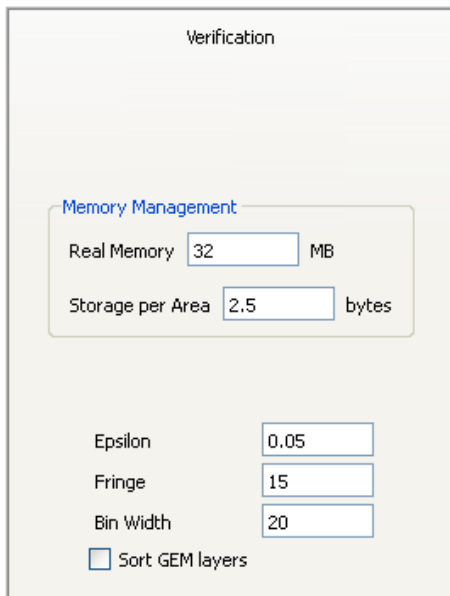
7. Click **Choose**.
8. In the Open Workspace window, click **Yes** to open the drc_via_wrk workspace, else click **No**.



After you click **Yes**, the drc_via_wrk example workspace opens in the selected directory.

Setting DRC Memory Use and Performance

To set the parameters that tune DRC's memory usage and performance, choose **Options > Preferences** in the Layout window and select the *Verification* tab.



Memory Management

The Memory Management area contains two fields:

- **Real Memory** is the real memory that DRC is allowed to use (in Mbyte). Adjust this value to a suitable value (for example, 250Mb) to avoid repeated memory allocations.
- **Storage per Area** is the amount of memory needed per unit area (in layout units). To enable a large design to be checked on a small machine, a design can be broken down into a list of smaller check regions called "facets". Using both Real Memory and Storage per Area factors, DRC decides whether the design fits in a single check region, or whether it requires a large number of smaller check regions:

$$\text{Maximum_check_area} = \text{real_memory_in_bytes} / \text{storage_per_area}$$
 You can usually specify a value of 2500 bytes in the *Storage per Area* field.

Epsilon

Epsilon is the offset to the clearance rule in DRC operation, to compensate for arithmetic rounding errors. Note this is in database units, not layout units. For example, for a 5 micron minimum spacing rule, this ensures that the edges, which are exactly 5 microns apart, will not be pulled in as an error.

For Example:

If `single_clearance (lyrCond) < 3.0` rule accidentally pulls in sections of edges which are exactly 3 units away due to rounding errors, in this case, epsilon REDUCES the clearance tested by a microscopic amount, so that it misses the offending edge.

Fringe

The fringe is used when the design must be broken down into facets due to memory constraints. In that case, the fringe value specifies to the program how facets must be expanded to catch errors that occur close to their borders.

The fringe is also used to define the upper bound value in 2-layer clearance tests with a GT or GE operator. For 1-layer clearance tests, it is recommended to specify a `DVE_RN_UPPER_BOUND` qualifier (refer to `DVE_RN_UPPER_BOUND (drc)` for more information).

The fringe is specified in layout units.

A fringe checking routine has been added to the rules compiler. It ensures that the fringe value is set internally to a value larger than the biggest rule or sizing operation. For example, the internal fringe for a width clearance rule is set at 2 times the width value. A message is provided if the fringe is set internally to a value larger than the value specified by the user:

```
Start rule compilation phase 1 ...
Using fringe value from largest rule: 359.800
Start rule compilation phase 2 ...
Rule compilation complete
```

The DRC engine checks errors between edges that are inside the facet and edges that are just outside the facet in the following way:

The border of the facet is called the "check zone".

```
##### <== check zone
#
#          *-----*   #   *-----*
#          |          |-->?<--|   |
#          |          |-->?<--|   |
#          |          |-->?<--|   |
#          *-----*   #   *-----*
#
#
#####
```

The DRC engine "bloats" the facet (the check zone) by the size of the fringe value. This forms a "rule zone" around the selected check zone.

Writing Design Rules

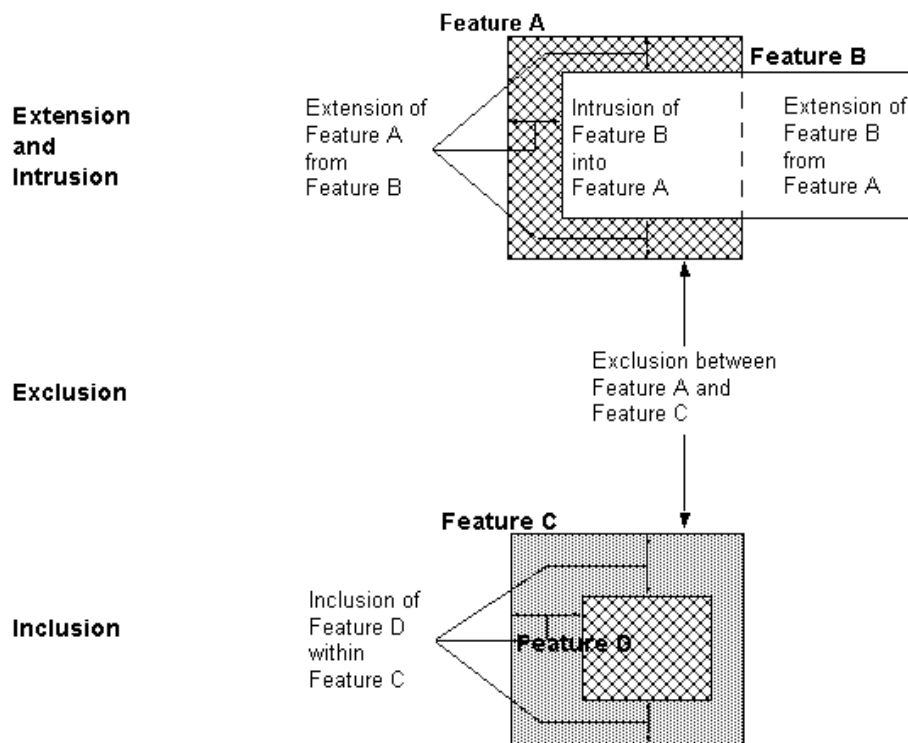
This section provides information for writing design rules. Design verification rules produce:

- graphical data showing the location of each violation
- an error message showing the nature of the violation.

A [complete DRC example](#) is included in this section. For detailed information on specific commands, see the command reference sections.

Extension and Intrusion Definitions

The terms, Extension and Intrusion, used in creating design rules, are defined in the following illustration.



Anatomy of a Simple DRC Rule File

A DRC rule file is written in Application Extension Language (AEL). The illustration shows a simple DRC rule file. Typically, a rule file consists of a Layer section and a Rule section. The Layer section declares all the design layers used or checked and all the output DRC layers for displaying errors. The Rule section consists of rule checking statements.

```

// ael rule file: subvia.ael
// Purpose: To check Via to Via spacing rule

// declare input design layers
decl backVia = dve_import_layer(20);

// declare output layer
decl lyrDRCErr = dve_export_layer(101);

//
// Substrate Via Spacing Design Rules
// Rule A - Substrate Via to Via minimum spacing 150 um
//

lyrDrcErr += dve_drc(gap(backVia) < 150,
                    "Substrate via edge to via edge min. is 150 um"
                    );

```

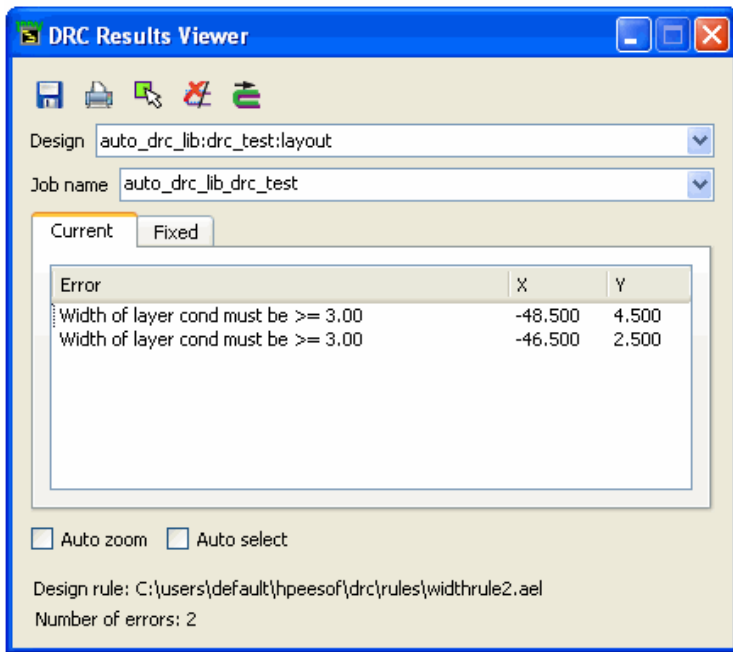
Note

A comment starts with a // or is enclosed by /* and */ .

Layer Management

The rules file illustrated in this section analyzes data on the physical design layer cond. The width command checks the inside clearance distance between edges of the same polygon. Edges that are less than 3.0 layout units apart are exported as line segments to design layer ads_drc_err. Each violation has an associated error message: width less than 3.0.

The AEL variable lyrCond references an import layer and the AEL variable drcError references an export layer.



Import Layers

When performing a design rule check, you must specify the design layers you want checked for design violations. Design layers from your layout design are imported into the verification process using the command *dve_import_layer* .

You can specify an import layer by using a layer name or a layer number:

```
decl lyrCond = dve_import_layer ("cond");
```

or

```
decl lyrCond = dve_import_layer (1);
```

Import layers can be used only as input to a DRC command. An import layer must be an existing *Physical* design layer and can only be used for import (that is, it cannot appear again on the left-hand side of a rule command).

Name	Number	Process Role	Binding
default	0	Not defined	
cond	1	Conductor	cond
cond2	2	Conductor	cond2
resi	3	Semiconductor	
diel	4	Dielectric	
diel2	12	Dielectric	
hole	5	Conductor Via	cond cond2
bond	6	Conductor	
symbol	7	Not defined	
text	8	Not defined	
leads	9	Not defined	
packages	10	Not defined	
ports	11	Not defined	
bound	13	Not defined	
silk_screen	14	Not defined	
silk_screen2	31	Not defined	
case_dimensions	15	Not defined	

Add Remove

Export Layers

Data is exported back to the layout editor by sending the output of the *dve_drc* command to an export layer. You create export layers using the command *dve_export_layer* .

You can specify an export layer by using a layer name or a layer number:

```
decl drcError = dve_export_layer ( " ads_drc_error " );
```

OR

```
decl drcError = dve_export_layer (101);
```

An export layer must be an existing DRC design layer. The Design Rule Checker will not display DRC errors on a Physical layer.

When sending a DRC error to an export layer, the += assignment is used to signify that you are performing an append operation. Export layers are always empty at the beginning of each DRC invocation, so it is safe to use the += append assignment when sending data to an export layer.

Export layers cannot be used as input to a DRC command. Export layers can appear only on the left-hand side of a rule command.

Work Layers

Work layers are used to reference intermediate data generated by a rule command. Work layers exist only temporarily while the DRC process is running, and are not part of the layout editor environment.

Use work layers when it is necessary to filter or process data on an import layer before generating a DRC error.

As good practice, you should always initialize a work layer to *NULL* .

Rules File Layers Example

This rules file example analyzes physical design data on layers cond and cond2. New polygons are created that represent the area where polygons on layer cond overlap polygons on layer cond2. The new polygons are placed in a work layer lyrPolyOverlap.

The *all_edges* command identifies the entire polygon as an error and the data is exported to DRC layer ads_drc_error.

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPolyOverlap = NULL;
lyrPolyOverlap = dve_bool_and (lyrCond, lyrCond2);
drcError += dve_drc (all_edges (lyrPolyOverlap),
"Conductive metal cond overlaps cond2");
```

Recognizing Devices in Flattened Layouts

ADS Release 2009 includes two new functions, *dve_identify_cell_layer()* (drc) and *de_touch()* (ael), that allow the DRC engine to recognize devices in flattened layouts. You can now write rules/commands for identifying the device/cell prior to running DRC rules and use the output layers of device recognition in the DRC engine for writing DRC rules. The *dve_identify_cell_layer()* command processes the design layers prior to input into DRC engine. It uses an associated AEL callback for the customization during the identification steps. The AEL command *de_touch ()* implements the touch-based device recognition rules.

Complete DRC Example

The example in this section illustrates writing design rules for Substrate Vias and NiCr Thin Film Resistors and manufacturing rules for Gate Metal. The example covers most of the functionalities and features of the DRC commands.

Note

The DRC file used in this example is included in the *drc_via_wrk* directory of the program's examples directory. For information on accessing the examples directory, see *Viewing DRC Examples*. (drc)

To set up a DRC check, you must define the design layers and the error layers. For information on setting up a DRC, refer to [Defining the Design Layers](#) and [Defining the Error Layers](#).

The following table shows the layer definitions for the process used in this example.

Mask Level	Layer	Description
Alignment Key	13	Defines fields in which alignment artifacts will be etched.
N+ Implant	2	Mask during alignment artifact etch, Implant mask for N+ regions.
D- Implant	1	Implant mask for DFET channels, Half DFET Diodes, D- Resistors.
NiCr	3	Liftoff layer for NiCr Resistors
Ohmic	5	Liftoff layer for ohmic contact on GaAs devices.
Isolation Implant	6	Implant mask for Isolation Implant
Gate Metal	7	Liftoff layer Schottky Gate/Anode contact on GaAs devices.
Metal 0	9	Liftoff layer for Metal 0
MIM	23	Liftoff layer for MIM metal
Via 1	14	First via etch layer
Metal 1	15	First plated Au metal layer. Labels are done in this layer
Air Bridge Post	10	Support Posts for Air Bridge and Via to Metal1
Air Bridge	11	Second plated Au metal layer
Passivation Via	12	Opens vias over bond pads and saw streets
Backside Via	20	Via holes (Via Option Only)
Backside Via Coat	21	Prevent solder wetting in vias (Via Option Only)

Defining the Design Layers

The rule section declares these imported design layers:

```
// declare input design layers
decl nImplant = dve_import_layer(2);
decl dImplant = dve_import_layer(1);
decl niCr = dve_import_layer(3);
decl ohmic = dve_import_layer(5);
decl isoImplant = dve_import_layer(6);
decl gateMetal = dve_import_layer(7);
decl metal0 = dve_import_layer(9);
decl mIM = dve_import_layer(23);
decl via1 = dve_import_layer(14);
decl metal1 = dve_import_layer(15);}}
decl airBridgePost = dve_import_layer(10);
decl airBridge = dve_import_layer(11);
decl passVia = dve_import_layer(12);
decl backVia = dve_import_layer(20);
decl backViaCoat = dve_import_layer(21);
```

Although every layer is declared here, you do not need to declare a design layer if you will not be checking it. This example does not use all of these layers, because you are not checking the complete design.

Defining the Error Layers

After defining the design layers, declare three DRC error layers to display errors from a set of rules. When writing DRC rules, you decide how many DRC error layers are needed to best view the results of a check.

```
// declare some DRC error layers
decl viaError = dve_export_layer(107); // for substrate via design rule
decl niCrError = dve_export_layer(103); // for thin film resistor rule
decl gateMetalError = dve_export_layer(120); // for gate metal rule
```


Note

DRC error message strings: () and _ are supported, but ` ' are not supported.

Checking the Clearance Rules

DRC checks clearance rules by selecting the edges that violate the clearance constraints and sending these to a DRC error layer. Clearance rules can be checked from either inside or outside of an edge to another edge of polygons.

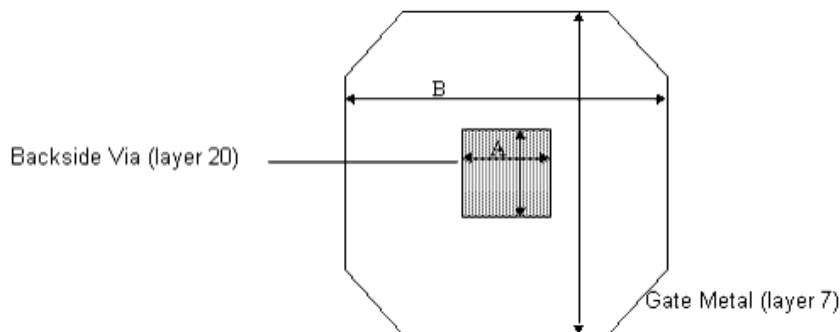
The types of clearance rules are:

- [width](#)
- [spacing](#)
- [external](#)
- [contains](#)
- [nests](#)
- [internal](#)

Of these, the simplest rule is width.

width

The width command is used to check the width of polygons on a given layer. The command checks the distance from the inside of one edge to the inside of another edge of the same polygon.



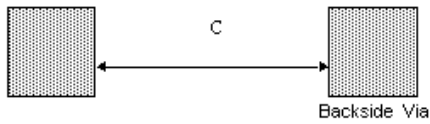
Item	Description	Minimum (um)
A	Coded Substrate Via Feature, Square (layer 20)	30
B	Substrate Via Target (layer 7)	120

Width rules for the substrate via are written as follows:

```
// Rule A: substrate via feature minimum 30 um
viaError += dve_drc(width(backVia) < 30, "Substrate via feature size < 30");
// Rule B: Substrate Via target size minimum 120 um
viaError += dve_drc(width(gateMetal) < 120, "Substrate via Target size < 120");
```

spacing

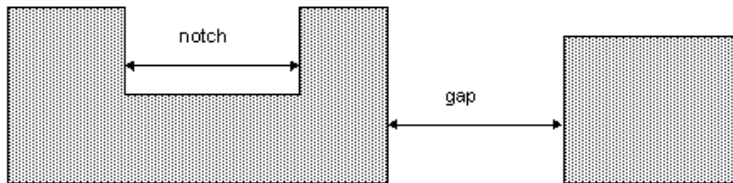
The spacing command is used to check spacing constraints on a given layer. The command checks the distance from the outside of an edge to the outside of another edge.



Item	Description	Minimum (um)
C	Substrate Via (layer 20) to Via (20), Edge to Edge	150

```
//
// Substrate Via Spacing Design Rule
// Rule C - Substrate Via to Via minimum spacing 150 um
//
viaError += dve_drc(spacing(backVia) < 150,
"Substrate via edge to via edge min. is 150 um");
```

Two other simple spacing rule commands are notch and gap. The notch command checks the spacing within the same polygon and the gap command checks the spacing between two different polygons. The spacing command checks both cases.

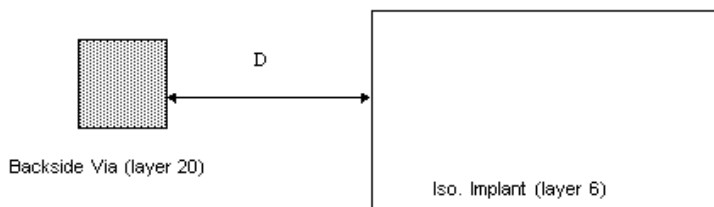


Checking Clearance Between Layers

All the clearance commands mentioned to this point work only on polygons that are on the same layer. Next you will see clearance commands that check the clearance from one layer to another. The layers checked can be a design or work layer, so you can send a design layer to a work layer and perform a two-layer rule command with the original design layer. An example of this capability is shown in the [Using Rule Conjunction](#).

external

The external command checks the external spacing between polygons on two different layers.

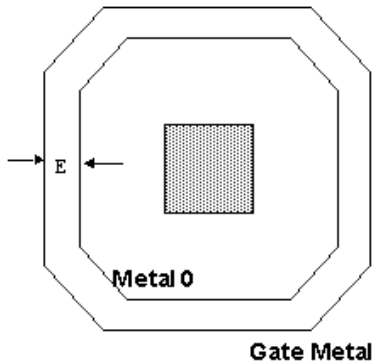


Item	Description	Minimum(um)
D	Substrate Via (layer 20) to Active Device Edge (layer 6)	90

```
//
// Rule D - Substrate Via to Iso. Implant minimum spacing 90 um
//
viaError += dve_drc(external(backVia, isoImplant) < 90,
"Substrate via edge to Iso. Implant Edge min. is 90 um");
```

contains

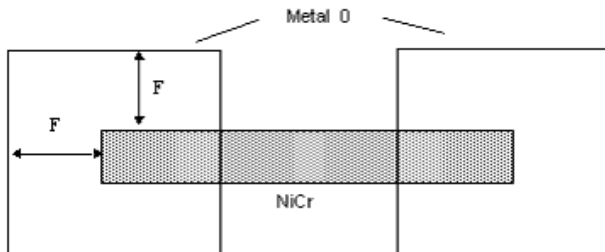
The contains command is used to check the inclusion of one polygon within another polygon. The command checks the distance from the inside edge of polygons on the first layer to the outside edge of polygons on the second layer.



Item	Description	Minimum(um)
E	Metal 0 (layer 9) Inclusion in Gate Metal (layer 7)	1.0

```
//
// Rule E - Metal 0 Inclusion in Gate Metal min is 1 um
gateMetalError += dve_drc(contains(gateMetal, metal0) < 1,
"Metal 0 Inclusion in Gate Metal min is 1 um");
```

You can use the contains command to check the extension of one polygon outside another polygon on a different layer. The illustration uses this design rule on NiCr Thin Film Resistors.



Item	Description	Minimum(um)
F	Metal 0 (layer 9) Extension from NiCr (layer 3)	0.5

```
// Rule F - Metal 0 Extension from NiCr min is 0.5 um
//
niCrError += dve_drc(contains(metal0, niCr) < 0.5,
"Metal 0 Extension from NiCr min is 0.5 um", DVE_RN_EDGE_ANGLES,
```

```
DVE_RV_PARALLEL);
```

nests

The nests command checks the distance from the outside edge of polygons on the first layer to the inside edge of polygons on the second layer. It is exactly the same command as the contains command except the two layer arguments are switched.

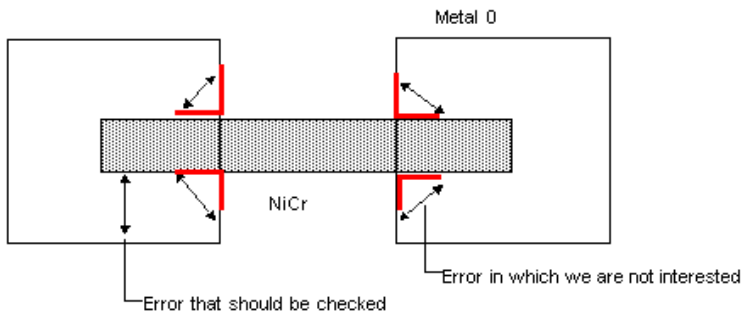
The example writes the previous extension rule (Rule F) using the nests command.

```
// Rule F - Metal 0 Extension from NiCr min is 0.5 um
//
niCrError += dve_drc(nests(niCr, metal0) < 0.5,
"Metal 0 Extension from NiCr min is 0.5 um",
DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL);
```

Notice that a qualifier was used in Rule F. A qualifier is defined as a name-and-value-pair:

Qualifier_Name, Qualifier_Value

Clearance Rule Qualifiers filter in (or out) tests between pairs of edges for a rule step. If no qualifier is specified, a rule command normally checks all the edge pairs. However, in this example, we are interested only in the edge pairs that are parallel to each other. Without the Parallel qualifier, we would get an unpleasant surprise from errors caused by non-parallel edges as shown in the following figure. Remember, *contains* checks from outside of the first polygon (on NiCr) to the inside of the second polygon (on Metal 0).



A width command appears to work well without a qualifier. What happens to the adjacent edges? Actually, the width command has a default qualifier to filter out all the adjacent edges during the rule operation:

DVE_RN_SEPARATE, DVE_RV_SEPARATE

Nearly all clearance commands have some type of default qualifiers to tell how the rule works. An example would be the Polarity qualifier. The fact that a command checks from the inside (or outside) of an edge to the inside (or outside) of another edge is dictated by the Polarity qualifier.

Two generic clearance commands (*single_clearance* and *double_clearance*) demand a polarity qualifier to tell them what to check. The *single_clearance* command is equivalent to a width command:

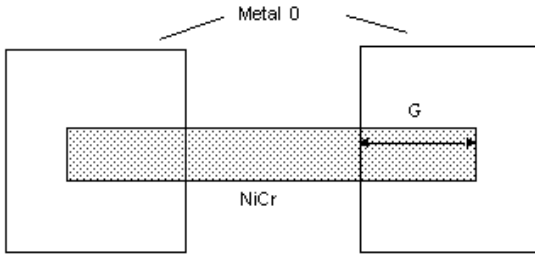
```
dve_drc(single_clearance(layer) < distance,
DVE_RN_POLARITY, DVE_RV_INSIDE);
```

The *double_clearance* command is equivalent to a contains command:

```
dve_drc(double_clearance(layer1, layer2) < distance,
DVE_RN_POLARITY_FROM, DVE_RV_INSIDE,
DVE_RN_POLARITY_TO, DVE_RV_OUTSIDE);
```

internal

This internal command checks the distance from the inside edge of one polygon to the inside edge of another polygon. The command is used to check the intrusion from one polygon into another polygon.



Item	Description	Minimum(um)
G	NiCr (layer 3) Intrusion into Metal 0 (layer 9)	2.5

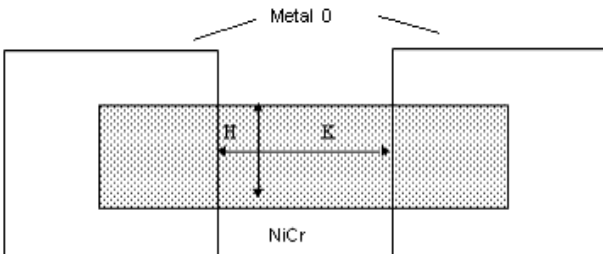
```
//
// NiCr Thin Film Design Rules
// Rule G - NiCr Intrusion into Metal0 min is 2.5 um
//
niCrError += dve_drc(internal(niCr, metal0) < 2.5,
"NiCr Intrusion into Metal0 min is 2.5 um");
```

Selecting Polygons

Several polygon selection commands are provided. In this example, only the `poly_path_length` and `poly_inter_layer` commands are described, but all polygon selection commands work similarly. For more details, see *Conditional Selection (drc)*.

poly_path_length

The command `poly_path_length` selects polygons based on the path length property of overlapping polygons on two layers.

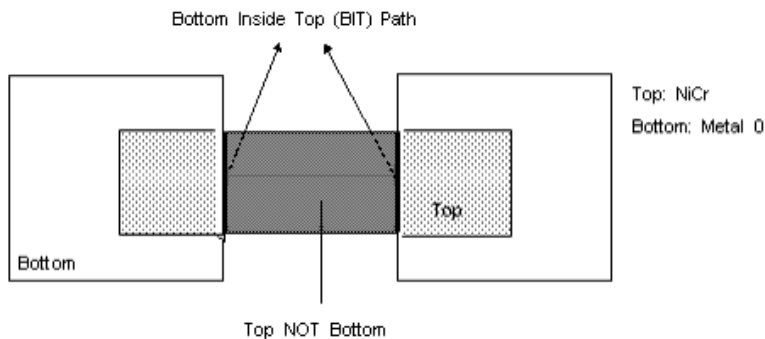


Item	Description	Minimum(um)
H	Resistor (layer 3) Width	2.0
K	Resistor (layer 3) Length	3.0

To check the width of a Thin Film Resistor, first do a boolean merge-NOT between the NiCr and Metal 0 layers to produce the resistor polygons. The path consisting of Bottom Inside Top (BIT) edges is the width of the resistor (see the illustration). Then select the bad resistors by checking the Bottom Inside Top (BIT) path length.

For details on determining the path code from merged polygons, refer to *Polygon Selection Based on Merge Properties*. (drc)

In this rule example, you begin to use work layers. Also, the result of a *poly_path_length* command is a polygon layer, so you need an *all_edges* command to send the polygon layer to a DRC error layer for displaying.



```
// declare some work layers
decl lyrResistor, widthShort;
//
// NiCr Thin Film Design Rules
// Rule H - Resistor width min is 2um
//
// To produce the resistor polygons
lyrResistor = dve_bool_not(niCr, metal0);
// Select if the BIT path length is less than 2
widthShort = dve_drc(poly_path_length(lyrResistor) < 2,
DVE_RN_PATH_CODE, DVE_RV_BIT, // set path code
DVE_RN_PATH_LENGTH, DVE_RV_MIN_PATH // check minimum
);
// Attach error message & send error polygons to DRC error layer
niCrError += dve_drc(all_edges(widthShort),
"NiCr Thin Film Resistor min width 2.0 um"
);
```

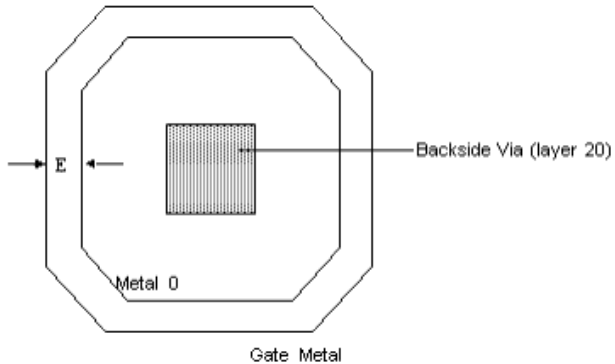
The Rule K checks the length dimension of a resistor. It does not require a *poly_path_length* command, you can implement this rule by using a boolean command and a clearance command. Try this as an exercise.

poly_inter_layer

The command *poly_inter_layer* selects a polygon based on its relationship to another polygon. The command is very useful for selecting a subset of polygons out of a polygon layer and then performing a rule check on the subset.

Item	Description	Minimum(um)
E	Metal 0 (layer 9) Inclusion in Gate Metal (layer 7)	1.0

Go back to rule *E*, which was done previously without filtering out unwanted polygons before applying the clearance command. This rule catches many errors that occur outside of substrate vias because both the Metal 0 and Gate Metal layers are used in the construction of other devices (such as DFET). The clearance rule brings in all of the polygons from these two layers, including the polygons used for DFET.



Fortunately, you can tell when a Metal 0 or a Gate Metal polygon is used for a substrate via: it must enclose a polygon from the Backside Via layer (layer 20), as shown on the illustration. The `poly_inter_layer` command is used to select polygons like this. Here is the rewritten Rule E:

```
//
// Substrate Via Spacing Design Rules
// Rule E - Metal 0 Inclusion in Gate Metal min is 1 um
//
// declare some work layers
decl viaGateMetal,viaMetal0; // these are work layers,
// that do not map to a real
// process layer
//
// First, derive gate metal used for substrate vias by using
// only the gate metal that encloses the backside via layer
//
viaGateMetal = dve_drc(poly_inter_layer(gateMetal, backVia),
DVE_RN_INTER_CODE, DVE_RV_ENCLOSE_ONLY);
//
// In a similar way, derive the metal0 used for substrate vias
//
viaMetal0 = dve_drc(poly_inter_layer(metal0, backVia),
DVE_RN_INTER_CODE, DVE_RV_ENCLOSE_ONLY);
//
// Use contains cmd to check Inclusion between 2 work layers
//
viaError += dve_drc(contains(viaGateMetal, viaMetal0) < 1,
"Metal 0 Inclusion in Gate Metal min is 1 um");
```

You can use the `poly_inter_layer` to detect whether two polygon layers overlap in a wrong manner. The command selects polygons by filtering in or out the overlapping conditions, such as Inside, Outside, Touch, and Cut, and then sends the polygons through an `all_edges` command to a DRC error layer. For more details, see `poly_inter_layer()`. (drc)

Using Rule Conjunction

In general, the result of deriving a work layer from one rule command and later feeding that work layer to another rule command is the combining of more than one rule constraint. This is called rule conjunction. In fact, you have seen rule conjunction in earlier examples of polygon selection commands. Here a more complicated example shows how

Advanced Design System 2011.01 - ADS Desktop Design Rule Checker
to use rule conjunction to check Gate Metal manufacturing rules.

Item	Description	Minimum (um)
L	Gate Metal (layer 7) spacing when width < 1.5	1.0

```
// declare output layer
decl gateMetalError = dve_export_layer(120);
//
// Gate Metal spacing Rule
// Rule L - Min. spacing is
// 1.0 if width < 1.5
// 1.5 if 1.5 <= width < 2.0
// 2.0 if 2.0 <= width < 3.0
// 3.0 if width >= 3.0
// declare some work layers
decl gatMet15Lt, gatMet15Ge, gatMet20Lt, gatMet20Ge;
decl gatMet30Lt, gatMet30Ge;
// Rule: Min. spacing is 1.0 if width < 1.5
// 1. select the edges with width < 1.5 from gateMetal, save in
// gatMet15Lt
// 2. select the edges with spacing error by checking the distance
// between gateMetal and gatMet15Lt
gatMet15Lt = dve_drc(width(gateMetal) < 1.5);
gateMetalError += dve_drc(external(gateMetal, gatMet15Lt) < 1.0,
"Gate Metal min spacing 1.0um when its width < 1.5um");
// Rule: Min. spacing is 1.5 if 1.5 <= width < 2.0
// 1. select the edges with width >= 1.5 from gateMetal, save in
// gatMet15Ge
// 2. select the edges with width < 2.0 from gatMet15Ge, save in
// gatMet20Lt
// 3. select the edges with spacing error by checking the distance
// between gateMetal and gatMet20Lt
gatMet15Ge = dve_drc(width(gateMetal) >= 1.5);
gatMet20Lt = dve_drc(width(gatMet15Ge) < 2.0);
gateMetalError += dve_drc(external(gateMetal, gatMet20Lt) < 1.5,
"Gate Metal min spacing 1.5um when its width within [1.5, 2)");
// Rule: Min. spacing is 2.0 if 2.0 <= width < 3.0
gatMet20Ge = dve_drc(width(gateMetal) >= 2.0);
gatMet30Lt = dve_drc(width(gatMet20Ge) < 3.0);
gateMetalError += dve_drc(external(gateMetal, gatMet30Lt) < 2.0,
"Gate Metal min spacing 2.0um when its width within [2.0, 3)");
// Rule: Min. spacing is 3.0 if width >= 3.0
gatMet30Ge = dve_drc(width(gateMetal) >= 3.0);
gateMetalError += dve_drc(external(gateMetal, gatMet30Ge) < 3.0,
"Gate Metal min spacing 3.0um when its width > 3.0 um");
```

Congratulations. You have finished writing your first rule file. If you would like to save it to a file, remember to use the file extension *.aef*. For details, see *Saving a DRC Rule. (drc)*

Additional DRC Examples

In addition to writing rules and checking for design errors, more complex tasks can be accomplished by means of additional AEL coding. The functionalities of AEL can be leveraged to develop highly customizable DRC solutions. The example below illustrates how to combine multiple DRC jobs together for a given design and then run all the jobs at once without further intervention.

Combining Multiple DRC Jobs Using AEL

The `dedrc_run_drc_ex` command enables combining multiple jobs. Consider the case below, where in multiple jobs for different rule sets viz., resistors, capacitors and other design rules, are to be combined in a single AEL script.

```
defun drc_run(sFileName)
```


Advanced Design System 2011.01 - ADS Desktop Design Rule Checker

```
{
// Get the currently active window
decl winInst = api_get_current_window();

// Check that the design loaded in the current window is valid and is a layout
decl context = de_get_current_design_context();
if( context == NULL || (!de_is_layout_context(context)) )
    return;
// DRC Job name
decl drcJobName = strcat(db_get_cell_name(context),"_drc");
// Run the job for rules
dedrc_run_drc_ex(winInst,sFilename,drcJobName,FALSE,FALSE);
// Unselect all items
de_deselect_all();
}
drc_run(strcat(getcwd(),"/auto_drc_rules.ael"));
```



Note

The AEL script is to be loaded from the ADS command line, to be executed. Optionally, this can be defined as a function and be invoked by linking to an user-defined menu item from ADS layout window. The AEL script should not be invoked from the DRC window.

DRC Functions (alphabetical)

[A](#) [C](#) [D](#) [E](#) [G](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#)

(For category-wise list of Functions, [click here \(drc\)](#))

A

Name	Type and Description
<i>all_edges()</i> (drc)	Boolean Operations on Edges. Sends all the edge segments of polygons of a layer to an output error layer.

C

Name	Type and Description
<i>compensate()</i> (drc)	Conditional Selection. Moves error segments on a given layer by a given distance. Output layer can only be used as input to <i>dve_quadout</i> and <i>dve_plgout</i> commands.
<i>contains()</i> (drc)	Conditional Selection. A DRC function to measure enclosure distance from the outside of the contained polygon to the inside of the containing polygon.
<i>corner edges()</i> (drc)	Conditional Selection. Generates error segments around corners of specified inside angles.

D

Advanced Design System 2011.01 - ADS Desktop Design Rule Checker

Name	Type and Description
<i>double clearance()</i> (drc)	Conditional Selection. Measures the distance between edges of polygons on different layers.
<i>de_touch()</i> (drc)	Selects polygons on one layer (inLayer1) in relation to edges of polygons on another layer (inLayer2) if TouchCondition is true. Output data to opLayer polygon layer.
<i>dve combine()</i> (drc)	Conditional Selection. Collects layers into one layer without modifying the shapes. Results of a combine operation can be used in edge and clearance rule operations. It is important to note that no merge or boolean operations are performed in the process.
<i>dve_bool_and()</i> (drc)	Merge Operations on Polygons. Merges overlapping polygons on two given layers.
<i>dve_bool_not()</i> (drc)	Merge Operations on Polygons. Subtracts shapes in the second layer from shapes in the first layer.
<i>dve_bool_or()</i> (drc)	Merge Operations on Polygons. Merges overlapping shapes on a given layer. Returns: A polygon layer.
<i>dve export layer()</i> (drc)	DRC Layer Management. Used to export DRC error information. Data written to an export layer will be directly exported back to the layout editor.
<i>dve drc()</i> (drc)	Conditional Selection. Used to select edges and polygons conditionally based upon intrinsic properties and information derived during an operation on one or more layers.
<i>dve drc group()</i> (drc)	Conditional Selection. Groups the layers and instructs DRC engine to generate a single DRC group. It loads the layers of a group only once instead of loading a layer for every layer.
<i>dve identify cell layer()</i> (drc)	DRC Layer Management. Processes the design layers prior to input into the DRC engine. Uses a AEL callback function for doing the customization in identification steps.
<i>dve import cell layer()</i> (drc)	DRC Layer Management. Used to get design data from the layout editor into the design verification process. Copies instance artwork from the layout editor onto an import layer that can be used in a rule command.
<i>dve import layer()</i> (drc)	DRC Layer Management. Used to get design data from the layout editor into the design verification process. Copies layer data from the layout editor onto an import layer that can be used in a rule command.
<i>dve_import_text_layer()</i> (drc)	DRC Layer Management. This command is useful for selecting polygons based on whether they contain certain text. It copies the layer data of texts into an import layer that can be used in a rule command and returns an import layer.
<i>dve oversize()</i> (drc)	Sizing Operations on Polygons. Moves edges of polygons by the given sizing distance. All edges are moved in parallel toward outside of polygons.
<i>dve_plgout()</i> (drc)	Operations for Polygon Extraction from Edges. Extracts entire polygons from selected edges. If any section of a polygon is in error, then the entire polygon is extracted.
<i>dve_quadout()</i> (drc)	Operations for Polygon Extraction from Edges. Extracts a quadrilateral from the selected error segments on the given layer.
<i>dedrc_run_drc_ex</i> (drc)	DRC Job Management. Used for executing DRC with an AEL command without invoking the DRC window. Can be combined with other AEL functions to develop customized a DRC solution.
<i>dve_segsize()</i> (drc)	Edge Selection Based on Corners. Expands or contracts an error segment on an edge. This command is particularly useful when used with the compensate command.
<i>dve undersize ()</i> (drc)	Sizing Operations on Polygons. Moves edges of polygons by the given sizing distance. All edges are moved in parallel toward inside of polygons.

E

Name	Type and Description
<i>external()</i> (drc)	Measures the distance between outside edges of polygons of different layers.
<i>Edge Qualifiers</i> (drc)	Use edge qualifiers either to select special options for a step of a rule or to filter tests between pairs of edges. These are called qualifiers because they qualify the rule.

G

Name	Type and Description
<i>gap()</i> (drc)	Conditional Selection. Measures the distance between outside edges of different polygons of the same layer.

I

Name	Type and Description
<i>invert_edges()</i> (drc)	Boolean Operations on Edges. Deselects selected edges and simultaneously selects unselected edges.
<i>internal()</i> (drc)	Measures clearance from the inside of one edge of a polygon to the inside of another edge of a different polygon.
<i>intrusion()</i> (drc)	Macros. Checks the intrusion of BOTTOM layer into TOP layer (see Convention for TOP and BOTTOM layers).

N

Name	Type and Description
<i>neests()</i> (drc)	Measures enclosure distance from the outside of the contained polygon to the inside of the containing polygon.
<i>notch()</i> (drc)	Conditional Selection. Measures the distance between outside edges of the same polygon on the given layer.

O

<i>off_grid()</i> (drc)	Flags edges whose end points fall off a specified grid.
-------------------------	---

P

Name	Type and Description
<i>poly_area()</i> (drc)	Polygon Selection. Selects polygons based upon area. For polygons with holes, the area of the hole is subtracted.
<i>poly_hole_count()</i> (drc)	Polygon Selection. Selects polygons based upon the number of holes.
<i>poly_line_length()</i> (drc)	Polygon Selection. Selects polygons based upon the minimum line length.
<i>poly_perimeter()</i> (drc)	Polygon Selection. Selects polygons based upon the total length of the outside edges.
<i>poly_edge_code()</i> (drc)	Polygon Selection. Select polygons based upon edge code information computed during a merge operation. Select only polygons with have all the given path types.
<i>poly_path_count()</i> (drc)	Polygon Selection. Select polygons based upon path count information computed during a merge operation.
<i>poly_path_length()</i> (drc)	Polygon Selection. Select polygons based upon path length properties computed during a merge operation.
<i>poly_inter_layer()</i> (drc)	Polygon Selection. Select polygons on one layer (inLayer1) in relation to edges of polygons on another layer (inLayer2) if any of the given constrains are true.
<i>protrusion()</i> (drc)	Checks the extension of TOP layer out of BOTTOM layer.

S

Name	Type and Description
<i>spacing()</i> (drc)	Conditional Selection. Simultaneously measures the distance between outside edges of different polygons of the same layer (<i>gap</i>) and outside edges of the same polygon (<i>notch</i>).
<i>single clearance()</i> (drc)	Conditional Selection. Measures the distance between edges of polygons on the same layer.

W

Name	Type and Description
<i>width()</i> (drc)	Conditional Selection. A DRC clearance function to check from the inside of one edge of a polygon to the inside of another edge of the same polygon.

DRC Functions (by category)

(For alphabetized list of Functions, [click here \(drc\)](#))

- [Import and Export Layers](#)
- [Edge Selection](#)
- [Edge Operations](#)
- [Polygon Selection](#)
- [Polygon Operations](#)
- [DRC Job Management](#)

Import and Export Layers

Name	Description
Importing Design Layers	
<i>dve import layer()</i> (drc)	Imports design data from the layout editor into the design verification process. Copies layer data from the layout editor onto an import layer that can be used in a rule command.
Exporting DRC Errors	
<i>dve_export_layer()</i> (drc)	Used to export DRC error information. Data written to an export layer will be directly exported back to the layout editor. Returns an export layer.
Cell Layers	
<i>dve identify cell layer()</i> (drc)	Processes the design layers prior to input into the DRC engine.
<i>dve import cell layer()</i> (drc)	Imports design data from the layout editor into the design verification process. Copies instance artwork from the layout editor onto an import layer that can be used in a rule command.
Text Layers	
<i>dve import text layer()</i> (drc)	Creates an import layer containing the bounding boxes of all text strings for the specified text. Can be used to create derived layers including or excluding other polygons overlaying these text markers.

Edge Selection

Name	Description
<i>dve_drc()</i> (drc)	Selects edges and polygons conditionally based upon intrinsic properties and information derived during an operation on one or more layers.
Edge Selection Based On Clearance (Between Polygons on a Single Layer)	
<i>width()</i> (drc)	A DRC clearance function to check from the inside of one edge of a polygon to the inside of another edge of the same polygon.
<i>gap()</i> (drc)	Measures the distance between outside edges of different polygons of the same layer.
<i>notch()</i> (drc)	Measures the distance between outside edges of the same polygon on the given layer.
<i>spacing()</i> (drc)	Simultaneously measures the distance between outside edges of different polygons of the same layer (<i>gap</i>) and outside edges of the same polygon (<i>notch</i>).
<i>single_clearance()</i> (drc)	Measures the distance between edges of polygons on the same layer.
Edge Selection Based On Clearance (Between Polygons on a Different Layers)	
<i>contains()</i> (drc)	A DRC function to measure enclosure distance from the outside of the contained polygon to the inside of the containing polygon.
<i>double_clearance()</i> (drc)	Measures the distance between edges of polygons on different layers.
<i>external()</i> (drc)	Measures the distance between outside edges of polygons of different layers.
<i>intrusion()</i> (drc)	Checks the intrusion of BOTTOM layer into TOP layer (see Convention for TOP and BOTTOM layers).
<i>internal()</i> (drc)	Measures clearance from the inside of one edge of a polygon to the inside of another edge of a different polygon.
<i>nests()</i> (drc)	Measures enclosure distance from the outside of the contained polygon to the inside of the containing polygon.
<i>protrusion()</i> (drc)	Checks the extension of TOP layer out of BOTTOM layer.
<i>Edge Qualifiers</i> (drc)	Use edge qualifiers either to select special options for a step of a rule or to filter tests between pairs of edges. These are called qualifiers because they qualify the rule.
Edge Selection Based on Corners	
<i>corner_edges()</i> (drc)	Generates error segments around corners of specified inside angles.
Edge Selection Based on Grid	
<i>off_grid()</i> (drc)	Flags edges whose end points fall off a specified grid.
Edge Selection at the Polygon Level	
<i>all_edges()</i> (drc)	Sends all the edge segments of polygons of a layer to an output error layer.
<i>invert_edges()</i> (drc)	Deselects selected edges and simultaneously selects unselected edges.

Edge Operations

Name	Description
Sizing Operations	
<i>compensate()</i> (drc)	Moves error segments on a given layer by a given distance.
<i>dve_segsize()</i> (drc)	Expands or contracts an error segment on an edge. This command is particularly useful when used with the <i>compensate</i> command.

Polygon Selection

Name	Description
Polygon Selection at the Layer Level	
<i>dve_combine()</i> (drc)	Collects layers into one layer without modifying the shapes.
<i>dve_drc_group()</i> (drc)	Groups the layers and instructs DRC engine to generate a single DRC group. It loads the layers of a group only once instead of loading a layer for every layer.
<i>de_touch()</i> (drc)	Selects polygons on one layer (inLayer1) in relation to edges of polygons on another layer (inLayer2) if TouchCondition is true. Output data to opLayer polygon layer.
Polygon Selection Based on Intrinsic Properties	
<i>poly_area()</i> (drc)	Selects polygons based upon area. For polygons with holes, the area of the hole is subtracted.
<i>poly_hole_count()</i> (drc)	Selects polygons based upon the number of holes.
<i>poly_line_length()</i> (drc)	Selects polygons based upon the minimum line length.
<i>poly_perimeter()</i> (drc)	Selects polygons based upon the total length of the outside edges.
Polygon Selection Based on Merge Properties	
<i>poly_edge_code()</i> (drc)	Select polygons based upon edge code information computed during a merge operation. Select only polygons with have all the given path types.
<i>poly_path_count()</i> (drc)	Select polygons based upon path count information computed during a merge operation.
<i>poly_path_length()</i> (drc)	Select polygons based upon path length properties computed during a merge operation.
Polygon Selection Based on Edge Relationships	
<i>poly_inter_layer()</i> (drc)	Select polygons on one layer (inLayer1) in relation to edges of polygons on another layer (inLayer2) if any of the given constrains are true.

Polygon Operations

Name	Description
Polygon Merge Operations	
<i>dve_bool_and()</i> (drc)	Merges overlapping polygons on two given layers.
<i>dve_bool_not()</i> (drc)	Subtracts shapes in the second layer from shapes in the first layer.
<i>dve_bool_or()</i> (drc)	Merges overlapping shapes on a given layer. Returns: A polygon layer.
Polygon Extraction Based on Selected Edges	
<i>dve_plgout()</i> (drc)	Extracts entire polygons from selected edges. If any section of a polygon is in error, then the entire polygon is extracted.
<i>dve_quadout()</i> (drc)	Extracts a quadrilateral from the selected error segments on the given layer.
Polygon Sizing Operations	
<i>dve_oversize()</i> (drc)	Moves edges of polygons by the given sizing distance. All edges are moved in parallel toward outside of polygons.
<i>dve_undersize()</i> (drc)	Moves edges of polygons by the given sizing distance. All edges are moved in parallel toward inside of polygons.

DRC Job Management

Name	Description
<i>dedrc_run_drc_ex</i> (drc)	Used for executing DRC with an AEL command without invoking the DRC window. Can be combined with other AEL functions to develop customized a DRC solution.

DRC Job Management

This section describes managing DRC using custom AEL scripting.

- *dedrc run drc ex()* (drc)

dedrc_run_drc_ex

Used for executing DRC with an AEL command without invoking the DRC window. Can be combined with other AEL functions to develop customized a DRC solution.

Syntax:

```
dedrc_run_drc_ex(winInst, rulesFile, jobName, checkWindowOnly, showResults);
```

where,

winInst is the *window* handle.

rulesFile is the *AEL rule filename* to be executed.

jobName is the name by which the *results are to be stored and displayed in the DRC results viewer window*.

checkWindowOnly indicates the *section of the layout to be considered for DRC*, where:

- FALSE = Includes the entire layout.
- TRUE = Includes the layout window currently in view.

showResults is used to *control the display of results after the DRC job is executed*, where:

- FALSE = Does not displays the results after executing the rules.
- TRUE = Invokes DRC results viewer to display the results after executing the rules.

Example

```
// get current active window
decl winInst = api_get_current_window();
// Declare the rule file name
decl ruleFileName = "my_rules.ael";
// Declare the job name for the DRC run
decl drcJobName = "my_rules_drc";
// Run the job for capacitor rules and show the results
dedrc_run_drc_ex(winInst, ruleFileName, drcJobName, FALSE, TRUE);
```


DRC Match Functions for Error Checking

This section describes how to manage DRC errors using custom AEL scripting. This feature provides powerful AEL function to match the error string of DRC error and modify errors. It enables the DRC engine to identify the errors and allow the AEL function to interpret the errors. Once the DRC runs, error checking is done in post processing mode.

You can also examine the DRC error in an AEL callback function and modify the error string, if needed. If the callback returns *NULL* the DRC error is deleted. This is useful for filtering out the false errors.

Match AEL Function

The Match AEL Function includes the following:

1. Register function
2. Error callback

Register Function

Syntax

```
de_add_error_callback (<DRC error message>,<error Call Back Name>)
```

Explanation

This function takes two arguments message to be changed and error call back name. DRC error callback is invoked for every occurrence of DRC error with the above message <DRC error message>.

Error Callback

Syntax

```
defun < error Call Back Name >( shapeH,drcID,drcMsg)
```

Complete Match AEL Function

```
defun drctest_cb(shapeH, drcID, drcMessage)
{
  decl newDrcMessage = NULL;
  decl length, width, ratio;
  decl airBridgeRatio = .5;
  decl dgBBox = db_get_shape_bbox(shapeH);
  decl x1 = db_get_bbox_x1(dgBBox);
  decl y1 = db_get_bbox_y1(dgBBox);
  decl x2 = db_get_bbox_x2(dgBBox);
  decl y2 = db_get_bbox_y2(dgBBox);
  length = x2 - x1;
  width = y2 - y1;
  if (width > 0)
  {
    ratio = length/width;
    if (ratio < airBridgeRatio)
    {
      newDrcMessage = strcat("Air bridge ratio less than ", identify_value(airBridgeRatio));
    }
  }
  return(newDrcMessage);
}
```

```

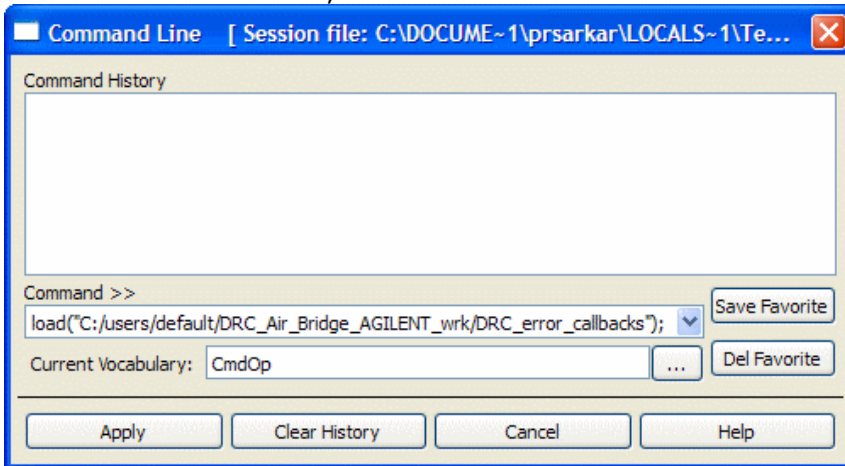
}
de_add_error_callback("air_bridge"," drctest_cb");

```

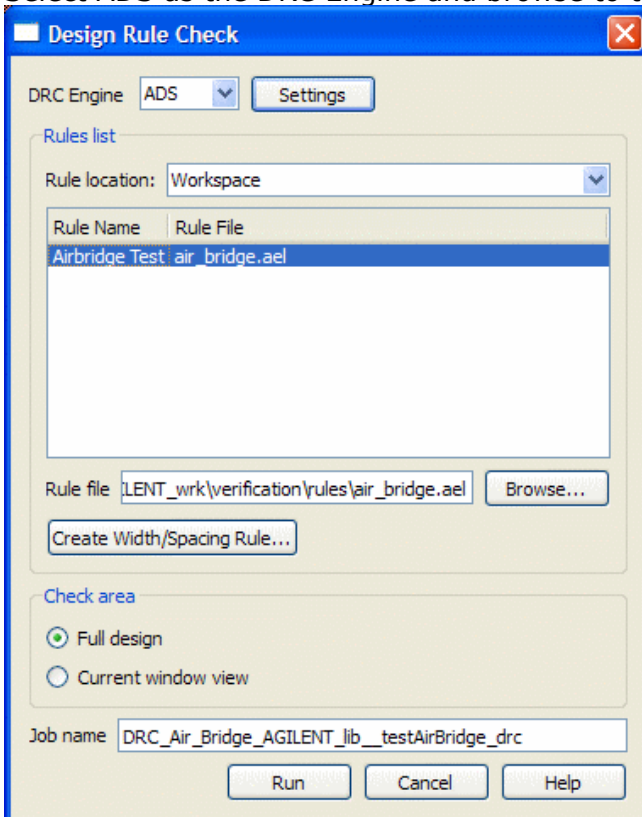
Running Match AEL function

To run Match AEL function:

1. From ADS Main window, click **Tools > Command Line** and load the Match AEL file.



2. Open a design in the Layout window.
3. From your Layout window, click Tools > DRC.
4. Select ADS as the DRC Engine and browse to the Rule file and click **Run**.



5. For further details on how to set up the DRC, refer *Setting Up a Quick DRC (drc)*.

Boolean Operations on Edges

This section describes the boolean operations between segments of an edge. These operations include:

- *all_edges()* (drc)
- *invert_edges()* (drc)

all_edges()

Sends all the edge segments of polygons of a layer to an output error layer.

See also: *dve_drc()* (drc)

Syntax

```
dve_drc (all_edges (inLayer) [, msgString]);
```

where:

<i>inLayer</i>	A polygon layer
<i>msgString</i>	A string value that will be attached to the selected error segments

Example

```
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrWork = NULL;
lyrWork = dve_drc (poly_area (lyrCond2) < 10.0);
drcError += dve_drc (all_edges (lyrWork),
    "Conductive metal area < 10.0")
```



invert_edges()

Deselects selected edges and simultaneously selects unselected edges.

See also: *dve_drc()* (*drc*)

Syntax

```
dve_drc (invert_edges (inLayer) [, msgString]);
```

where:

<i>inLayer</i>	A polygon layer
<i>msgString</i>	A string value that will be attached to the selected error segments

Example

The following example demonstrates the use of *invert_edges()* to highlight paths that are within a specified clearance distance.

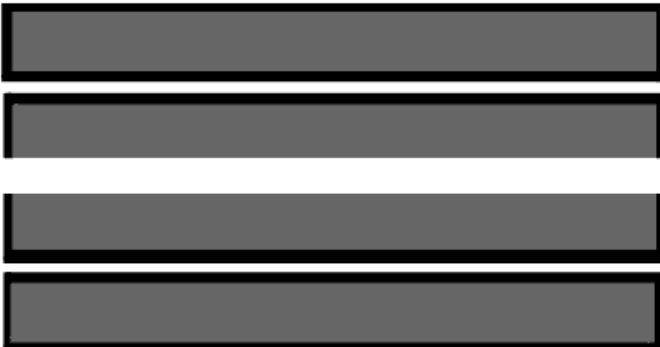
First, measure the distance between parallel edges and select edges that are within the distance of 12.

```
decl lyrCond = dve_import_layer ("cond");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrEdgesGap = dve_drc (single_clearance (lyrCond) <= 12.0,
DVE_RN_POLARITY, DVE_RV_OUTSIDE, DVE_RN_TEMPLATE, DVE_RV_OPPOSITE,
DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL, DVE_RN_STRUCTURE, DVE_RV_DIFF_POLYGON);
```



Next, use *invert_edges* to invert the error segments: good become bad and bad become good.

```
decl lyrEdgesInvert= dve_drc (invert_edges (lyrEdgesGap));
```



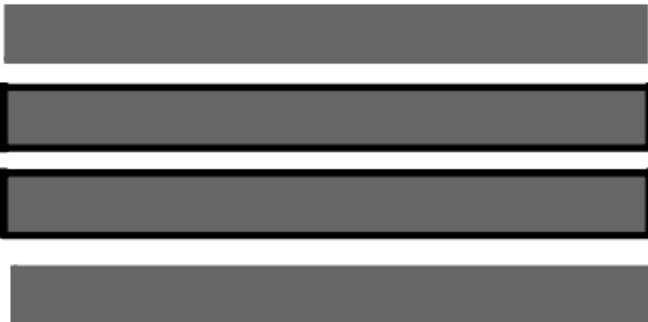
Select opposite edges inside the paths:

```
decl lyrEdges= dve_drc (double_clearance (lyrEdgesGap, lyrEdgesInvert) 26.0,  
  DVE_RN_POLARITY, DVE_RV_INSIDE,  
  DVE_RN_TEMPLATE, DVE_RV_OPPOSITE,  
  DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL);
```



Finally, extract the path using quadout:

```
decl lyrPoly = dve_quadout (lyrEdges);  
drcError = dve_drc (all_edges (lyrPoly), "Parallel interconnect < 12.0");
```



Conditional Selection

This section describes the DRC commands used for conditional selection. The section includes information on:

- *dve_drc()* (drc)
- *dve_combine()* (drc)
- *dve_drc_group()* (drc)
- *de_touch()* (drc)
- *Edge Selection Based On Clearance* (drc)
 - *width()* (drc)
 - *gap()* (drc)
 - *notch()* (drc)
 - *spacing()* (drc)
 - *single_clearance()* (drc)
 - *internal()* (drc)
 - *external()* (drc)
 - *contains()* (drc)
 - *nests()* (drc)
 - *double_clearance()* (drc)
 - *Edge Qualifiers* (drc)
- *Edge Selection Based on Corners* (drc)
 - *corner_edges()* (drc)
- *Edge Selection Based on Grid* (drc)
 - *off_grid()* (drc)
- *Edge Compensation* (drc)
 - *compensate()* (drc)
 - *dve_segsize()* (drc)

compensate()

Moves error segments on a given layer by a given distance. Output layer can only be used as input to *dve_quadout* and *dve_plgout* commands. Returns: A layer with selected edge segments.

See also: *dve_plgout()* (drc), *dve_quadout()* (drc)

Syntax

```
edgeLayerOut = dve_drc(compensate (edgeLayerIn, distance [,resourceName, resourceValue]);
```

where:

<i>edgeLayerIn, edgeLayerOut</i>	An edge layer
<i>distance</i>	A real value

Compensate Template Qualifier

DVE_RN_COMP_TEMPLATE

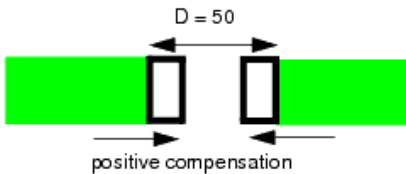
Qualifier Resource Value

<i>DVE_RV_CHAMFER</i>	Compensate using an angle from the orthogonal
<i>DVE_RV_ALIGN</i>	(default) Compensate using an alignment to the adjacent edge
<i>DVE_RV_BISECT</i>	Compensate where the angle is bisected at the corner
<i>DVE_RV_OPPOSITE</i>	Compensate directly opposite the edge

Positive and Negative Compensations

Both positive compensation and negative compensation are supported.

Positive compensation: for example, suppose that when two edges face one another within a specified clearance distance $D=50$ they must be compensated towards each other by 10.0



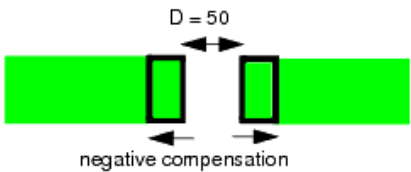
```
lyrEdges = dve_drc(spacing(lyrCond) < 50, "d < 50", DVE_RN_EDGE_ANGLES,
DVE_RV_PARALLEL);
```

```
lyrEdgesComp = dve_drc(compensate(lyrEdges, 10));
```

```
lyrPolyComp = dve_quadout(lyrEdgesComp);
```

```
drcError = dve_drc(all_edges(lyrPolyComp), "positive compensation");
```

For negative compensation, edges are moved away from each other by the specified amount:



Then a not function can be used to cut these sections away from the original polygon:



```
lyrEdges = dve_drc(spacing(lyrCond) < 50, "d < 50", DVE_RN_EDGE_ANGLES,
DVE_RV_PARALLEL);
```

```
lyrEdgesComp = dve_drc(compensate(lyrEdges, -10));
```

```
lyrPolyComp = dve_quadout(lyrEdgesComp);
```

```
lyrPolyNot = dve_bool_not(lyrCond, lyrPolyComp);
```

Adjusting the COMPENSATE command

The compensate command can be given qualifiers which tell it:

- How to chamfer a compensated segment.
- How to specify the behavior at a concave and a convex corner.

How to chamfer a compensated segment

Normally, the compensated segment is projected orthogonally to the edge:



As an alternative, specify a chamfer angle in degrees:

chamfer angle

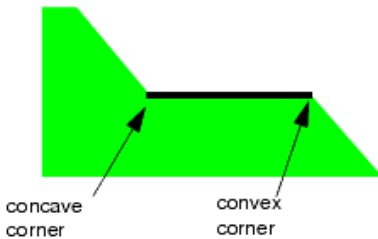


```
lyrEdgesComp = dve_drc(compensate(lyrEdges ,30), DVE_RN_COMP_TEMPLATE,
    DVE_RV_CHAMFER, DVE_RN_CHAMFER_ANGLE, 45);
```

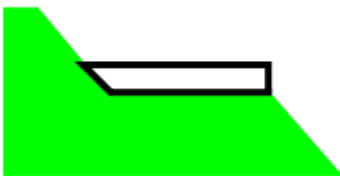
The chamfer angle is expressed as the deviation from the orthogonal, so "DVE_RN_CHAMFER_ANGLE, 0" is the (default) orthogonal.

The template used by the compensate command

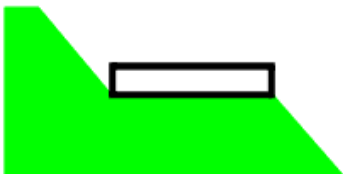
Consider an edge to be compensated, which ends at a concave and a convex corner:



The default behavior is to align the compensated section at the concave corner and to project it orthogonally at the convex corner:

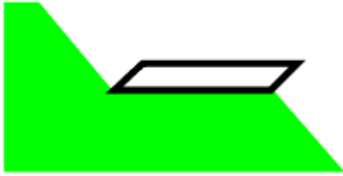


A different behavior can be defined by specifying a compensate template:
Compensate with an opposite template



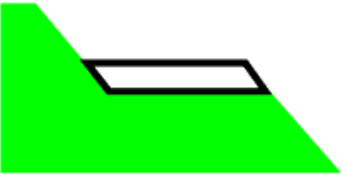
```
lyrEdgesComp = dve_drc(compensate(lyrEdges, 5), DVE_RN_COMP_TEMPLATE, DVE_RV_OPPOSITE);
```

Compensate with a bisect template. The program bisects the angle at the corners:



```
lyrEdgesComp = dve_drc(compensate(lyrEdges ,5), DVE_RN_COMP_TEMPLATE, DVE_RV_BISECT);
```

Compensate with an align template. The compensated section is aligned at the adjacent edge:



```
lyrEdgesComp = dve_drc(compensate(lyrEdges ,5), DVE_RN_COMP_TEMPLATE, DVE_RV_ALIGN);
```

Example

```
decl lyrCond = dve_import_layer ("cond");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrEdges = NULL;
decl lyrEdgesComp = NULL;
decl lyrPolyCond = NULL;
decl lyrPolyComp = NULL;
decl lyrPolyOversize = NULL;
// Generate an oversized polygon
lyrEdges = dve_drc (width (lyrCond) < 5.0);
lyrEdgesComp = dve_drc (compensate (lyrEdges, 0.5),
                        DVE_RN_COMP_TEMPLATE, DVE_RV_CHAMFER,
                        DVE_RN_CHAMFER_ANGLE, 45);
lyrPolyCond = dve_quadout (lyrEdges);
lyrPolyComp = dve_quadout (lyrEdgesComp);
lyrPolyOversize = dve_bool_or (lyrPolyCond, lyrPolyComp);
// Check gap clearance
drcError += dve_drc (gap (lyrPolyOversize) < 4.0, "Gap clearance < 4.0");
```

contains()

A DRC function to measure enclosure distance from the outside of the contained polygon to the inside of the containing polygon.

See also: *dve_drc()* (*drc*)

Syntax

```
dve_drc (contains (inLayer1, inLayer2) operator distance [, msgString]
[, qualifierName, qualifierValue...]);
```

where:

<i>inLayer1</i>	Containing polygon layer
<i>inLayer2</i>	Contained polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the selected error segments
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

Edge Qualifiers

DVE_RN_EDGE_ANGLES (drc)

DVE_RN_ANGLE_TOLERANCE (drc)

DVE_RN_SEPARATE (drc)

DVE_RN_TOUCH (drc)

DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO (drc)

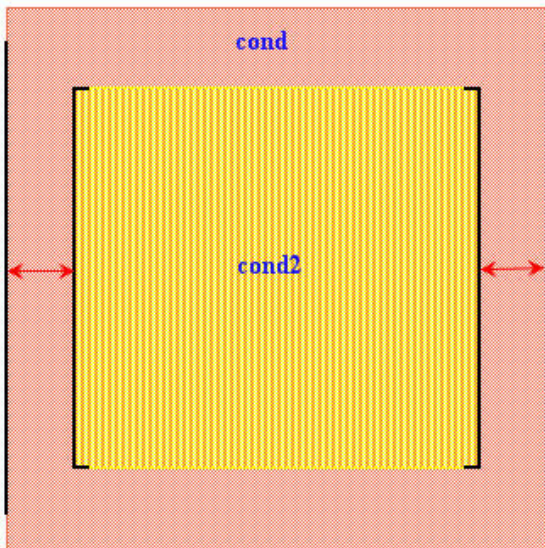
DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM (drc)

DVE_RN_UPPER_BOUND (drc)

i The *contains()* command checks the similar edges that are checked by the *nests()* (drc) command. The only difference between them being the order of layers passed.

Example

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
drcError += dve_drc (contains (lyrCond, lyrCond2) < 3.0, "Enclosure clearance < 3.0");
```

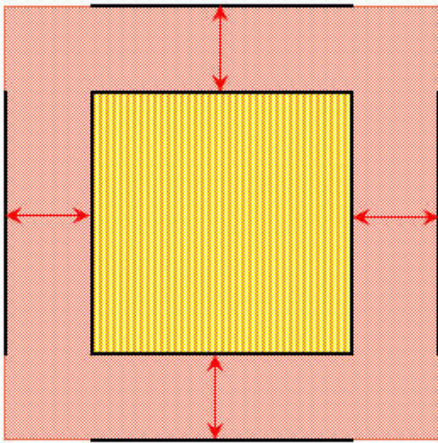
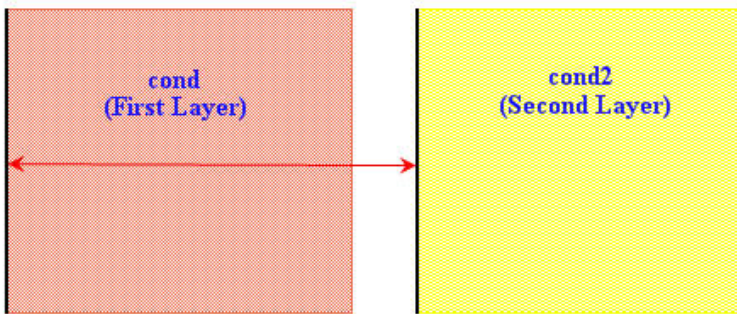


i The order of the layers is important while using the *contains()* command. First layer is the containing layer and second layer is the contained layer. By default, the intersecting edges are also checked. To ignore it, use the qualifier *DVE_RN_SEPARATE* (drc) with the resource value 'DVE_RV_SEPARATE'

Note
 The `contains()` command will also check on the edges even when the two different layers polygons are not included in each other if the clearance distance satisfies the constraint. A user may see this behavior as a false error but actually the DRC engine is flagging the error when the constraint is satisfied. The DRC engine will not check which is a containing or a contained layer.

Constraint is **'from Inside Edge of First Layer to Outside Edge of Second Layer'**,

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
// cond is the First Layer and cond2 is the Second Layer
drcError += dve_drc (contains (lyrCond, lyrCond2) < 3.0,
  "Enclosure clearance < 3.0",
  DVE_RN_TEMPLATE,DVE_RV_OPPOSITE);
```



Tip
 To avoid a situation as shown above, declare a work layer which has a Boolean AND operation of the two layers in consideration. The code may be modified as:

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl temp1 = dve_bool_and(lyrCond2,lyrCond);
drcError += dve_drc (contains(lyrCond,temp1) < 3.0,
  "Enclosure clearance < 3.0",
  DVE_RN_TEMPLATE,DVE_RV_OPPOSITE);
```

corner_edges()

Generates error segments around corners of specified inside angles.

See also: `dve_drc()` (`drc`)

Syntax

`dve_drc` (`corner_edges` (`inLayer`, `segmentLength`, `beginningAngle`, `endingAngle`) [`,msgString`]);

where:

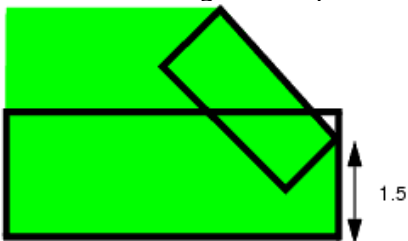
<code>inLayer</code>	A polygon layer
<code>segmentLength</code>	A real value in layout units that represents the length of the error segment that will be drawn around the corner
<code>beginningAngle</code>	A real value that represents the minimum angle that will be selected
<code>endingAngle</code>	A real value that represents the max angle that will be selected
<code>msgString</code>	A string value that will be attached to the selected error segments

Example 1

```
decl lyrCond = dve_import_layer ("cond");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrEdgesCvex = NULL;
decl lyrEdgesStub = NULL;
decl lyrStub = NULL;
lyrEdgesCvex = dve_drc (corner_edges (lyrCond, 0.5, 1, 91));
lyrEdgesStub = dve_drc (single_clearance (lyrEdgesCvex) < 3.0,
    DVE_RN_POLARITY, DVE_RV_INSIDE,
    DVE_RN_TEMPLATE, DVE_RV_OPPOSITE,
    DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL,
    DVE_RN_STRUCTURE, DVE_RV_SAME_POLYGON);
lyrStub = dve_quadout (lyrEdgesStub);
drcError += dve_drc (all_edges (lyrStub), "Stub");
```

Example 2

Consider some geometry with a chamfer corner and a rule to check the width of 2.0:



```
drc_error += dve_drc(width(cond) < 2, "Conductor width less than 2.0 um");
```

This rule fails to detect the error because the default template for `width()` is `DVE_RV_OPPOSITE`. The rectangular opposite template from the bottom edge hits the sloping edge, but the template from the sloping edge misses the bottom edge. We can change this by using an arc template with a specified curvature. This does detect the error but also has false errors at the top. The solution is to restrict the test to act between

- an orthogonal (90 degrees) corner
- an obtuse (> 90 degrees < 180 degrees) corner
so that we miss the false errors between pairs of obtuse corners. We can pick out

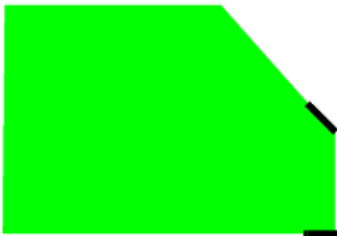
these corners with the `corner_edges` command:

```
a_orth= dve_drc(corner_edges(lyrCond, 0.2, 89.9, 90.1), "orthogonal corner");
a_obtuse= dve_drc(corner_edges(lyrCond, 0.2, 90.1, 179.9), "obtuse corner");
```

Note
The program only holds angles to 0.1 degree precision. Also, we don't test the orthogonal corner for exactly 90 degrees, because this might fail if the whole geometry was at a different angle.

Then we apply the equivalent of a width test from the orthogonal to obtuse corners. This is done using a `double_clearance` rule, from the inside of each edge at the orthogonal and obtuse corners:

```
drc_error += dve_drc(double_clearance(a_orth, a_obtuse) < 2.0, "test width from orthogonal to obtuse corners", DVE_RN_POLARITY_FROM, DVE_RV_INSIDE, DVE_RN_POLARITY_TO, DVE_RV_INSIDE, DVE_RN_TEMPLATE_FROM, DVE_RV_OPPOSITE, DVE_RN_TEMPLATE_TO, DVE_RV_ARC_OPPOSITE, DVE_RN_SEPARATE, DVE_RV_SEPARATE);
```



Note
The qualifier `DVE_RV_SEPARATE` is used to apply the test only to non-intersecting edges.

Edge Selection Based on Grid

Edge Selection Based on Grid selection function includes: `off_grid()` (drc).

de_touch()

Selects polygons on one layer (`inLayer1`) in relation to edges of polygons on another layer (`inLayer2`) if `TouchCondition` is true. Output data to `opLayer` polygon layer.

Syntax:

```
de_touch(inlayer1, inlayer2, opLayer, ACCEPT/REJECT, "TouchCondition")
```

where

<code>inLayer1</code>	Input layer 1
<code>inLayer2</code>	Input layer 2
<code>opLayer</code>	Output layer
<code>ACCEPT</code>	Selects the matching polygons
<code>REJECTS</code>	Does not select the matching polygons
<code>TouchCondition</code>	Can be one of these - ">n" or "<n" or "=n" or "!n" where n is a +ve integer

Example

```
// The below code creates a copy of cond layer polygons
// which touch more than one cond2 layer polygon into the temp layer.
de_touch( cond, cond2, temp, ACCEPT, >1 );
// The below code creates a copy of cond layer polygons
// which DONOT touch more than one cond2 layer polygon into the temp layer.
de_touch( cond, cond2, temp, REJECT, >1 );
```

i This ADS AEL function is to be used inside the callback associated with one of the DRC import layer functions. It is not to be used directly in the rule file.

double_clearance()

Measures the distance between edges of polygons on different layers.

See also: *dve_drc()* (drc)

Syntax

dve_drc (double_clearance (inLayer1, inLayer2) operator distance [, msgString] [,qualifierName, qualifierValue...]);

where:

<i>inLayer1</i>	Containing polygon layer
<i>inLayer2</i>	Contained polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the selected error segments
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

Edge Qualifier

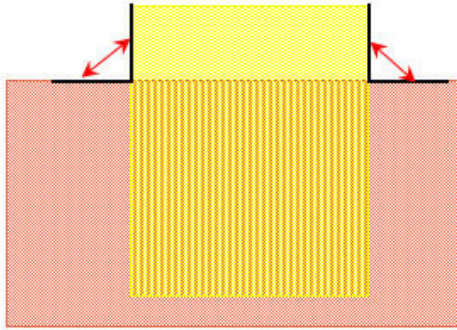
DVE_RN_POLARITY, DVE_RN_POLARITY_FROM, DVE_RN_POLARITY_TO (drc)
DVE_RN_EDGE_ANGLES (drc)
DVE_RN_ANGLE_TOLERANCE (drc)
DVE_RN_SEPARATE (drc)
DVE_RN_TOUCH (drc)
DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO (drc)
DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM (drc)
DVE_RN_UPPER_BOUND (drc)

Examples

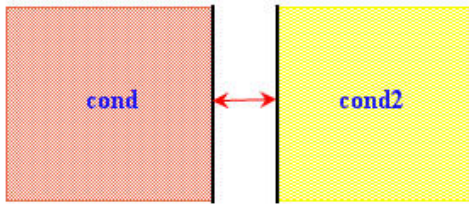
1. Double clearance check without any *Edge Qualifiers* (drc).

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_cond2 = dve_import_layer("cond2");
decl lyr_error = dve_export_layer(101);
```

```
lyr_error += dve_drc(double_clearance(lyr_cond, lyr_cond2) < 3.00,
    "Outside edges separation < 3.0");
```



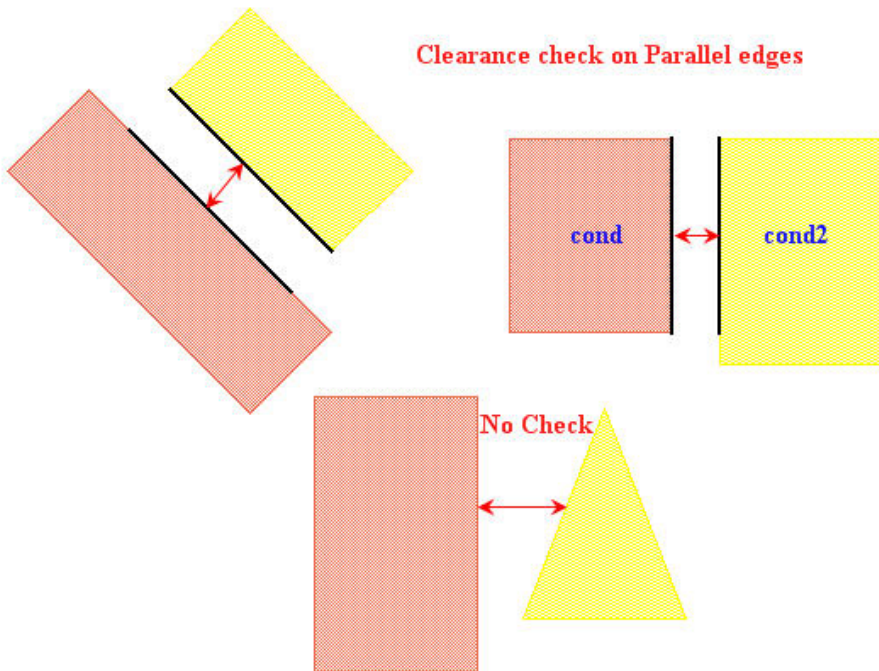
clearance check on outside edges of different polygons on different layers



i By default, *double_clearance()* will check on the outside edges

Double clearance check on parallel edges.

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_cond2 = dve_import_layer("cond2");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(double_clearance(lyr_cond, lyr_cond2) < 3.00,
    "Outside edges parallel separation < 3.0",
    DVE_RN_EDGE_ANGLES,DVE_RV_PARALLEL);
```

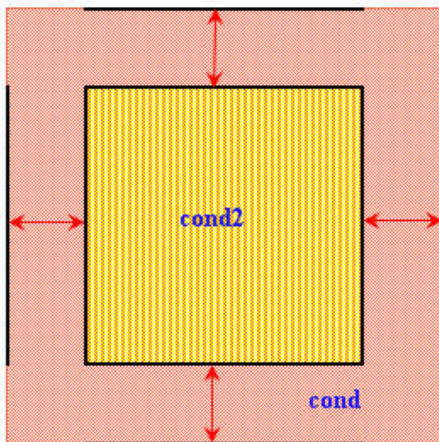


Double clearance check from inside edge of first layer to outside edge of second layer.

```

decl lyr_cond = dve_import_layer("cond");
decl lyr_cond2 = dve_import_layer("cond2");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(double_clearance(lyr_cond, lyr_cond2) < 3.00,
    "cond2 inside cond clearance < 3.0",
    DVE_RN_POLARITY_FROM,DVE_RV_INSIDE,
    DVE_RN_POLARITY_TO,DVE_RV_OUTSIDE);

```



i Here, cond is the first layer and cond2 is the second layer as passed as inLayers in *double_clearance()*. If the order is reversed then cond2 will become the first layer and there will be no error in the figure shown as Polarity FROM is INSIDE edge of the first layer

dve_combine()

Collects layers into one layer without modifying the shapes. Results of a combine operation can be used in edge and clearance rule operations. It is important to note that no merge or boolean operations are performed in the process. Returns: a polygon layer.

Syntax

```
dve_combine ( inLayer1 [, inLayer2, . . . , inLayerN])
```

where:

<i>inLayer1, inLayer2, inLayerN</i>	A polygon layer
-------------------------------------	-----------------

Example

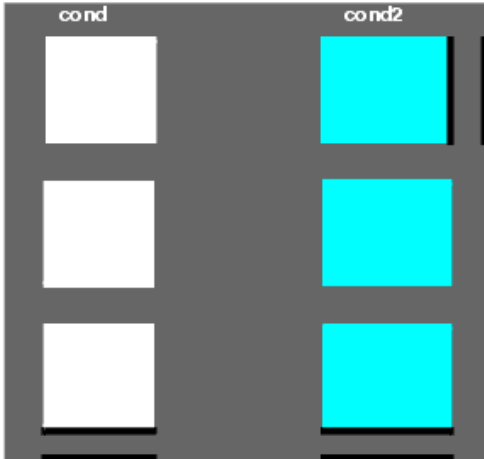
```

decl lyrCond = dve_import_layer("cond");
decl lyrCond2 = dve_import_layer("cond2");
decl lyrDiel = dve_import_layer("diel");
decl drcError = dve_export_layer("ads_drc_error");
// collect layers cond and cond2 into the same layer so that shapes
// on these layers are checked together
decl lyrPolyOverlap = dve_combine(lyrCond ,lyrCond2);
// apply clearance rule

```



```
drcError += dve_drc(double_clearance(lyrDiel,lyrPolyOverlap) < 5,
"Conductive metal overlaps", DVE_RN_POLARITY_FROM, DVE_RV_INSIDE,
DVE_RN_POLARITY_TO, DVE_RV_OUTSIDE);
```



Edge Selection Based On Clearance

The Edge Selection Based On Clearance selection functions are used where the output layer contains polygons with selected edges. Functions are separated by number of layers.

1 Layer Check:

- *width()* (drc) checks between inside edges of the same polygon
- *gap()* (drc) checks between outside edges of different polygons
- *notch()* (drc) checks between outside edges of the same polygon
- *spacing()* (drc) combines gap and notch tests
- *single_clearance()* (drc) "neutral command", no polarity (outside, inside) and no qualifier ("same" or "different polygons") is specified

2 Layer Check:

- *internal()* (drc) the "width" command for 2 layers
- *external()* (drc) the "gap" command for 2 layers
- *contains()* (drc) checks from the inside edges on the first layer to the outside edges on the second layer
- *nests()* (drc) checks from the outside edges on the first layer to the inside edges on the second layer
- *double_clearance()* (drc) "neutral command", no polarity (outside, inside) and no qualifier ("same" or "different polygons") is specified

These are the default behaviors. The default can be changed by specifying:

- A polarity (see *DVE_RN_POLARITY*, *DVE_RN_POLARITY_FROM*, *DVE_RN_POLARITY_TO* (drc))
- A qualifier such as *DVE_RN_STRUCTURE* (drc)

Note

Be careful when changing the default behavior. For example if you specify "DVE_RN_POLARITY_FROM, DVE_RV_OUTSIDE, DVE_RN_POLARITY_TO, DVE_RV_OUTSIDE" for a width command, it actually checks notch!

dve_drc()

Used to select edges and polygons conditionally based upon intrinsic properties and information derived during an operation on one or more layers. Returns: a layer containing selected edge segments.

Syntax

```
dve_drc (drc_expression [, msgString][, qualifierName, qualifierValue]);
```

where:

<i>drc_expression</i>	is an AEL expression in the format:
<i>drc_subfunction</i>	A selection function to be performed on the polygons or edges on a given layer. Edges and polygons that meet the criteria are selected and copied to the output layer. The subfunctions are: <i>Edge Selection Based On Clearance</i> (drc) (output layer contains polygons with selected edges) selection functions are separated by number of layers: <i>1 Layer check</i> : gap, notch, single_clearance, spacing, width <i>2 Layer check</i> : contains, double_clearance, external, internal, nests, <i>Edge Selection Based on Corners</i> (drc) selection functions include: corner_edges <i>Edge Selection Based on Grid</i> (drc) selection functions include: off_grid <i>Edge Compensation</i> (drc) selection functions include: compensate, dve_segsize <i>Polygon Selection Based on Intrinsic Properties</i> (drc) (output layer contains polygons) selection functions include: poly_area, poly_hole_count, poly_line_length, poly_perimeter <i>Polygon Selection Based on Merge Properties</i> (drc) (output layer contains polygons) selection functions include: poly_edge_code, poly_path_count, poly_path_length <i>Polygon Selection Based on Edge Relationships</i> (drc) (output layer contains polygons) selection functions include: poly_inter_layer
<i>parameter</i>	A parameter to a dve_drc subfunction command
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>rValue</i>	A real or integer value that depends upon the DRC subfunction
<i>msgString</i>	A string that will be attached to the selected edges. Only pertains to selected edges. Can only be used in conjunction with the export nomenclature (such as, "+=")
<i>qualifierName</i>	A constant that represents the name of the qualifier. Use qualifiers to select special options of a rule, or to filter tests between a pair of edges. Qualifiers are documented for each dve_drc subfunction
<i>qualifierValue</i>	A value that will be applied to the named qualifier. Valid range of values are documented for each dve_drc subfunction

Example

```
decl lyrCond = dve_import_layer ("cond");
decl drcError = dve_export_layer ("ads_drc_error");
drcError += dve_drc (width (lyrCond) < 3.0, "Width of conductive metal < 3.0");
```

dve_drc_group()

Groups the layers and instructs DRC engine to generate a single DRC group. It loads the layers of a group only once instead of loading a layer for every layer. In the below example, layers lyrCond, lyrCond2, lyrDie will be loaded only once. Note: DRC engine

might run out of memory if too many layers are loaded in a group. If DRC engine runs out of memory then a message is displayed in the status window. To overcome this problem, it is recommended to reduce the number of layers in a group and define multiple groups.

Syntax

```
dve_drc_group ( inLayer1 [, inLayer2, . . ., inLayerN])
```

where:

<i>inLayer1, inLayer2, inLayerN</i>	A polygon layer
-------------------------------------	-----------------

Example

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrDiel = dve_import_layer ("diel");
decl lyrResi = dve_import_layer ("resi");
decl drcError = dve_export_layer ("ads_drc_error");
// Group layers lyrCond, lyrCond2 and lyrDiel
dve_drc_group(lyrCond, lyrCond2, lyrDiel);
drcError += dve_drc(external(lyrCond,lyrCond2)<25,"Minimum spacing b/w layer cond and cond2 is 25");
drcError += dve_drc(external(lyrCond,lyrDiel)<15,"Minimum spacing b/w layer cond and diel is 15");
drcError += dve_drc(external(lyrCond2,lyrDiel)<25,"Minimum spacing b/w layer cond2 and diel is 25");
// Group layers lyrCond, lyrCond2 and lyrResi
dve_drc_group(lyrCond, lyrCond2, lyrResi);
drcError += dve_bool_and(lyrCond,lyrCond2);
drcError += dve_bool_and(lyrCond,lyrResi);
drcError += dve_bool_and(lyrCond2,lyrResi);
```

dve_segsiz()

Expands or contracts an error segment on an edge. This command is particularly useful when used with the `compensate` command.

See also: `compensate()` (drc).

Syntax

```
edgeLayerOut = dve_segsiz(errorEdgeLayer, distance);
```

where

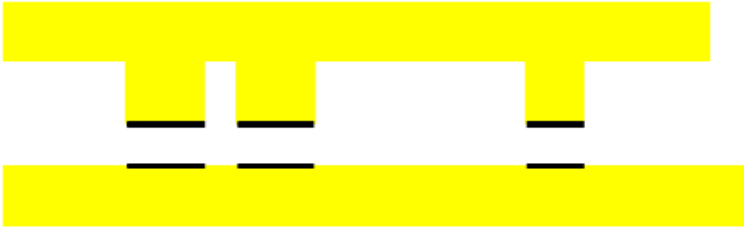
<i>edgeLayerOut</i>	An edge layer
<i>errorEdgeLayer</i>	A layer with error segments
<i>distance</i>	A real value

For a positive distance the expansion stops at a vertex. For a negative distance a segment that ends at a vertex remains "pinned" to that vertex.

Example

A requirement to compensate error segments, based on some clearance operation:

```
decl lyrMetal = dve_import_layer("Metal1");
decl drcError = dve_export_layer("Error101");
drcError= dve_drc(single_clearance(lyrMetal ) < 20.0, DVE_RN_EDGE_ANGLES,
  DVE_RV_PARALLEL, DVE_RN_POLARITY, DVE_RV_OUTSIDE, DVE_RN_STRUCTURE,
  DVE_RV_DIFF_POLYGON, DVE_RN_TEMPLATE, DVE_RV_OPPOSITE);
```



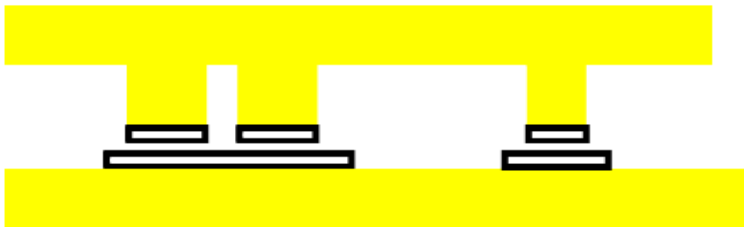
Now, perform compensation:

```
decl lyrEdges, lyrEdgesCmp, lyrPolyCmp;
lyrEdges = dve_drc(single_clearance(lyrMetal ) < 20.0, ...);
lyrEdgesCmp = dve_drc(compensate(lyrEdges, 4));
lyrPolyCmp = dve_quadout(lyrEdgesCmp);
drcError += dve_drc(all_edges(lyrPolyCmp), "clearance < 20");
```



This may produce undesirably narrow notches between the sections on the left. Eliminate these notches with the segsize operation, which expands or contracts every error segments by a specified amount:

```
decl lyrEdges, lyrEdgesSized, lyrEdgesCmp, lyrPolyCmp;
lyrEdges = dve_drc(single_clearance(lyrMetal ) < 20.0,...);
lyrEdgesSized = dve_segsize(lyrEdges, 20 );
lyrEdgesCmp = dve_drc(compensate(lyrEdgesSized, 4));
lyrPolyCmp = dve_quadout(lyrEdgesCmp);
drcError += dve_drc(all_edges(lyrPolyCmp), "clearance < 20");
```



Note that the results of the segsize operation merge together. Now, perform a segsize with a negative distance:

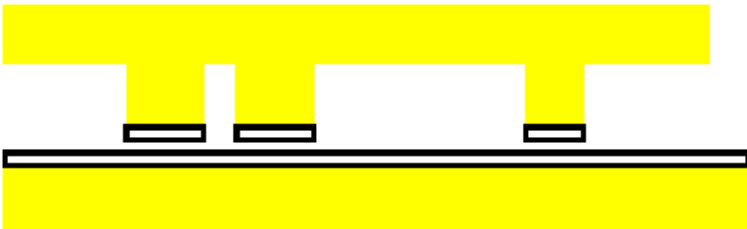
```
decl lyrEdges, lyrEdgesSized, lyrEdgesUnderSized, lyrEdgesCmp, lyrPolyCmp;
lyrEdges = dve_drc(single_clearance(lyrMetal ) < 20.0,...);
lyrEdgesSized = dve_segsize(lyrEdges , 20 );
lyrEdgesUnderSized = dve_segsize(lyrEdgesSized , -20 );
lyrEdgesCmp = dve_drc(compensate(lyrEdgesUnderSized , 4));
```

```
lyrPolyCmp = dve_quadout(lyrEdgesCmp );
drcError += dve_drc(all_edges(lyrPolyCmp), "clearance < 20");
```



Edges that contain error segments can be extracted by specifying a distance larger than the edge length (edgeout operation):

```
lyrEdgesSized = dve_segsize(lyrEdges, 500);
```



Edge Qualifiers

Use edge qualifiers either to select special options for a step of a rule or to filter tests between pairs of edges. These are called qualifiers because they qualify the rule.

DVE_RN_UPPER_BOUND

Specifies the upper bound value in a 1-layer clearance test with a GT or GE operator.

Qualifier Resource Value:

This qualifier applies to any conditional-selection command that uses the "Greater than" GT or "Greater than or equal to n" GE operators. The upper bound is often used to specify a range for the following commands: *gap()* (drc), *notch()* (drc), *spacing()* (drc), *width()* (drc) and *single_clearance()* (drc).

The fringe value is used as the upper bound value if this qualifier is not specified in a GT or GE clearance test (see *dveFringe* under *Preference File Format and Descriptions* (custom) in the *Customization and Configuration* (custom) documentation).

Example:

```
// bounded test
drc_error += dve_drc(gap(cond) > 2, DVE_RN_UPPER_BOUND, 7,
    DVE_RN_TEMPLATE, DVE_RV_OPPOSITE, "2 < gap <= 7");
// use the fringe value
drc_error += dve_drc(gap(cond) > 2, DVE_RN_TEMPLATE,
    DVE_RV_OPPOSITE, "2 < gap <= fringe");
```

DVE_RN_EDGE_ANGLES

Qualifier Resource Value:

<i>DVE_RV_PARALLEL</i>	Select only parallel edges
<i>DVE_RV_NOT_PARALLEL</i>	Select only non-parallel edges
<i>DVE_RV_PERPENDICULAR</i>	Select only perpendicular edges
<i>DVE_RV_NOT_PERPENDICULAR</i>	Select only non-perpendicular edges
<i>DVE_RV_ANY_ANGLE</i>	(<i>default</i>) Select edges at any angle

Note

DVE_RV_PARALLEL and *DVE_RV_PERPENDICULAR* are mutually exclusive.
DVE_RV_NOT_PARALLEL and *DVE_RV_NOT_PERPENDICULAR* are not mutually exclusive.

DVE_RN_ANGLE_TOLERANCE

Qualifier Resource Value:

< <i>real value</i> >	Edge angle tolerance in degrees
-----------------------	---------------------------------

This qualifier can only be used in conjunction with *RUL_RN_EDGE_ANGLES*.

For example:

```
text += dve_drc(external(cond2, cond) < 1.0,
  "cond2 separation from cond < 1.0 um",
  DVE_RN_EDGE_ANGLES, DVE_RV_NOT_PARALLEL,
  DVE_RN_ANGLE_TOLERANCE, 10.0);
```

Using *DVE_RN_ANGLE_TOLERANCE* without specifying an angle qualifier will result in a warning: qualifier ignored.

DVE_RN_POLARITY, DVE_RN_POLARITY_FROM, DVE_RN_POLARITY_TO

Qualifier Resource Value:

<i>DVE_RV_INDSIDE</i>	Direct search toward inside of polygon
<i>DVE_RV_OUTSIDE</i>	(<i>default</i>) Direct search toward outside of polygon

DVE_RN_STRUCTURE

Qualifier Resource Value:

<i>DVE_RV_ANY_POLYGON</i>	(<i>default</i>) Test applies to any edge
<i>DVE_RV_SAME_POLYGON</i>	Test applies only between edge of same polygon
<i>DVE_RV_DIFF_POLYGON</i>	Test applies only between edge of different polygons

DVE_RN_SEPARATE

Determines how two adjacent edges are checked.

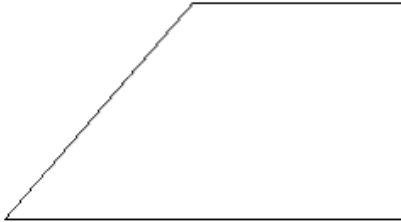
Qualifier Resource Value:

<i>DVE_RV_SEPARATE</i>	applies only to non-intersecting edges
<i>DVE_RV_NOT_SEPARATE</i>	applies only to intersecting edges
<i>DVE_RV_ANY_SEPARATE</i>	applies to edges regardless of whether they intersect or not
<i>DVE_RV_PERP_SEPARATE</i>	applies only if two adjacent edges are not perpendicular
<i>DVE_RV_JOIN_SEPARATE</i>	applies only to non-intersecting edges, joining edges are added in

DVE_RV_SEPARATE normally applies to a width or a notch test, so that an edge is not checked against its immediate neighbors in a polygon

Examples

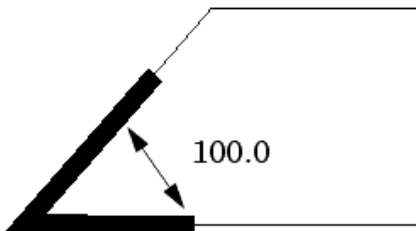
1. Consider some geometry with an acute angle and a rule to check the width of 100:



```
drcError = dve_drc(width(lyrEdges) < 100.0);
```

This rule fails to detect the error at the acute angle. Use the following rule conjunction to address this problem:

```
// insert error segments onto the edges forming an acute angle
lyrEdges= dve_drc(corner_edges(lyrCond, 200.0, 0.1, 89.9));
// show the width which is in error. This can be done by applying a width
// check. Enable DVE_RV_ANY_SEPARATE, so that adjacent edges are checked.
drcError = dve_drc(width(lyrEdges) < 100.0, DVE_RN_SEPARATE,
DVE_RV_ANY_SEPARATE);
```



2. See the command *corner_edges()* (drc) for an example with *DVE_RV_SEPARATE*.

DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO

Use a slope qualifier to activate a rule step only if it has a specified slope.

Qualifier Resource Value:

<i>DVE_RV_VERTICAL</i>	Select only vertical edges
<i>DVE_RV_HORIZONTAL</i>	Select only horizontal edges
<i>DVE_RV_ORTHOGONAL</i>	Select only vertical and horizontal edges
<i>DVE_RV_DIAGONAL</i>	Select only diagonal edges
<i>DVE_RV_OCTAGONAL</i>	Select only vertical, horizontal and diagonal edges
<i>DVE_RV_OTHER</i>	Select only non-octagonal edges
<i>DVE_RV_LEFT</i>	Select only the left edge
<i>DVE_RV_BOTTOM</i>	Select only the bottom edge
<i>DVE_RV_RIGHT</i>	Select only the right edge
<i>DVE_RV_TOP</i>	Select only the top edge
<i>DVE_RV_ALL_SLOPES</i>	(default) Select edges at any slope

Examples

1. Consider a gap check which is applied only between vertical edges:



```
drcError= dve_drc(gap(1yrCond) < 35.0, DVE_RN_SLOPE, DVE_RV_VERTICAL);
```

2. Consider a gap check which is applied only between diagonal edges:



```
drcError= dve_drc(gap(1yrCond) < 35.0, DVE_RN_SLOPE, DVE_RV_DIAGONAL);
```

3. Consider a gap check which is applied only between orthogonal and diagonal edges:



```
drcError= dve_drc(gap(1yrCond) < 35.0, DVE_RN_SLOPE_FROM,  
DVE_RV_ORTHOGONAL, DVE_RN_SLOPE_TO, DVE_RV_DIAGONAL);
```


DVE_RN_TOUCH

Qualifier Resource Value:

DVE_RV_DTOUCH	Edges which touch are also diagnosed as errors. The error segment is constrained to the touching edges.
DVE_RV_CLEAR_TOUCH	Edges which touch are also diagnosed as errors. The error segment extends beyond the touching edges.
DVE_RV_OVERLAP	Edges which overlap are also diagnosed as errors.

Examples

1. Consider two polygons on layer lyrCond and lyrCond2 which touch externally and an external rule:



```
drcError = dve_drc(external(lyrCond,lyrCond2) < 30.0, "d min is 30",
DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL);
```

The default is that butting edges are not diagnosed as errors. Now consider the use of the qualifier DVE_RV_CLEAR_TOUCH:

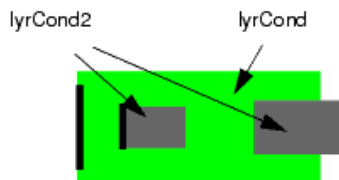
```
drcError = dve_drc(external(lyrCond,lyrCond2) < 30.0, "d min is 30",
DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL, DVE_RN_TOUCH, DVE_RV_CLEAR_TOUCH);
```

Then the touching section is given as an error:



2. Consider some geometry on 2 layers and a nests rule:

```
drcError = dve_drc(nests(lyrCond2,lyrCond) < 30.0, "d min is 30",
DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL);
```



```
drcError = dve_drc(nests(lyrCond2,lyrCond) < 30.0, "d min is 30",
DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL, DVE_RN_TOUCH, DVE_RV_OVERLAP);
```

detects violations of "nests" distance, and also diagnoses edges of polygons on layer lyrCond2 which are outside of polygons on layer lyrCond1:



Similarly,

```
drcError = dve_drc(external(lyrCond2,lyrCond) < 30.0, "d min is 30",
DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL, DVE_RN_TOUCH, DVE_RV_OVERLAP);
```

detects violations of "external" distance, and also diagnoses edges of polygons on layer lyrCond2 which are inside of polygons on layer lyrCond1:

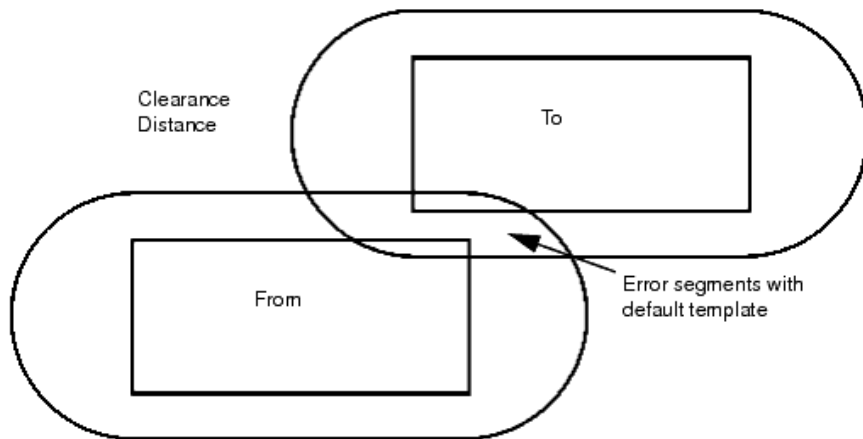


DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM

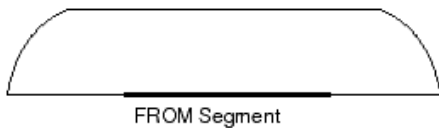
Control over templates is very important. Most false errors or missed real errors can be eliminated with carefully specified templates. The program starts with very pessimistic templates, usually round ones, which may generate false errors. Specifying a particular template may eliminate these errors.

How Clearance Templates are Applied

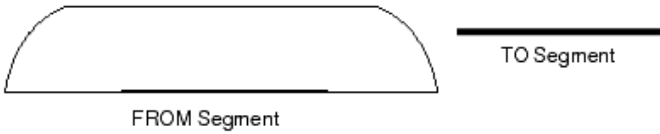
Clearance checks are done between edges of polygons, referred to as FROM and TO.



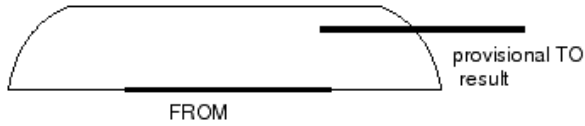
A template is constructed around each FROM edge.



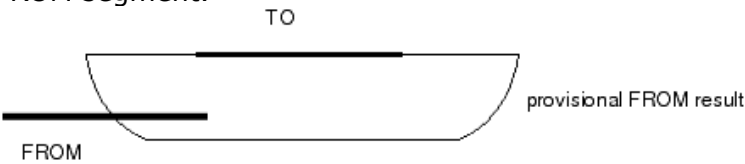
The program checks if this template captures a TO segment. If no edge crosses this template, it is regarded as a "miss" between these FROM and TO segments, and no more checks are made between them. Below is an example of a miss:



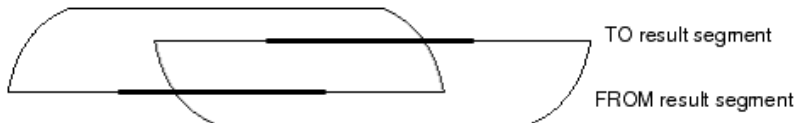
But if there is a "hit" with a To segment, the program creates a provisional result segment consisting of the parts of the TO edge which are within the FROM template.



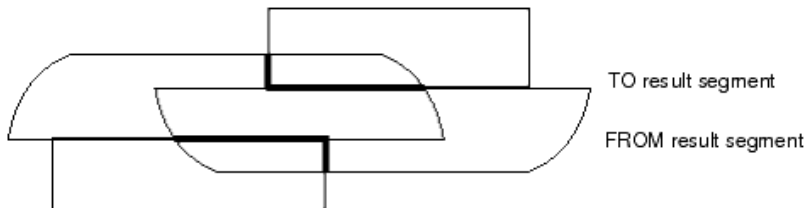
Then the process is reversed and a template is constructed on the TO segment (not just from the TO result segment), and the program checks to see if the template encloses a FROM segment.



If there is a hit on this second pass, the provisional segments are accepted for both the FROM and the TO tests, and they are added as result segments.



Also, if this was the last rule of a conjunctive set, the program relates the new result error segments. In this case, edges adjacent to those shown are also checked so the error segments usually go around corners.

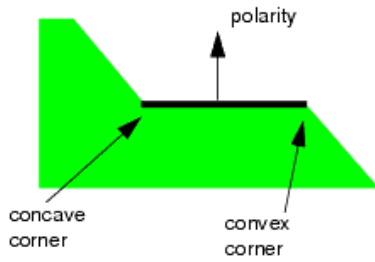


The length of the error segment around the corner acts as a visual clue to the severity of the error.

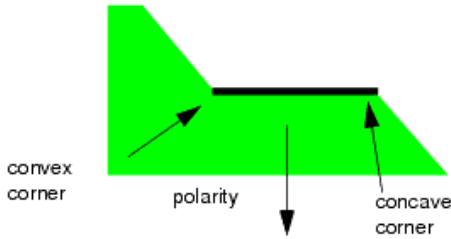
Specifying a Template

A template for a rule is defined by specifying the shape that is applied for a concave corner of the polygon and for a convex corner of the polygon.

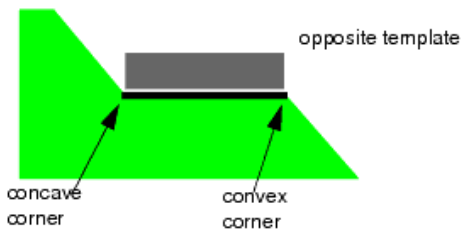
Consider an edge that has a concave corner at one end, and a convex corner at the other end:



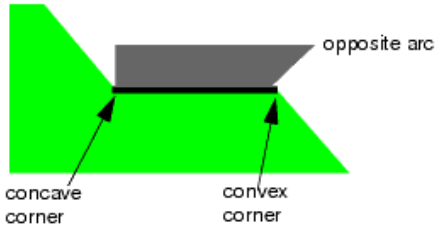
A concave corner subtends an angle of less than 180 degrees when looking from the edge in the direction of the polarity. A convex corner subtends an angle of more than 180 degrees. If the polarity of the rule is reversed, then the concave and convex corners are also reversed:



The same template can be applied to both ends of the line. For example:



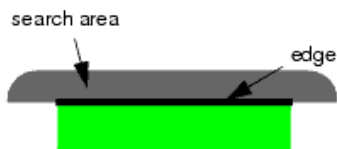
Or a different template can be specified for the concave corner and the convex corner:



Types of Templates

The choices for each end of the edge are:

- round - Extend search area using rounded corners



- opposite - (default) Extend search area just opposite the edge



- arc - Extend search area using arced corners. Refer to note below.

**Note**

The Arc template requires a curvature angle. Curvature is expressed as the angle (in degrees) by which the arc is raised. An Arc with a curvature angle of 0 degrees is equivalent to the "round" template; and Arc with a curvature angle of 90 degrees is equivalent to the "opposite" template.

- square - Extend search area treating corners as squares



or for checking both sides of a line, the template

- bothsides - Must be used for both convex and concave corners. Extend search area on both sides of edge. Use this template with the polarity DVE_RN_POLARITY



Specify whether the template is to be used for the FROM segment, the TO segment or both segments. Refer, to [How Clearance Templates are Applied](#) for the definition of FROM and TO and refer to [Specifying a Template](#) for the definition of concave and convex corners.

Qualifier Resource Values:

DVE_RV_ROUND	DVE_RV_ARC
DVE_RV_ROUND_ARC	DVE_RV_ARC_OPPOSITE
DVE_RV_ROUND_OPPOSITE	DVE_RV_ARC_ROUND
DVE_RV_ROUND_SQUARE	DVE_RV_ARC_SQUARE
DVE_RV_OPPOSITE	DVE_RV_SQUARE
DVE_RV_OPPOSITE_ARC	DVE_RV_SQUARE_ARC
DVE_RV_OPPOSITE_ROUND	DVE_RV_SQUARE_OPPOSITE
DVE_RV_OPPOSITE_SQUARE	DVE_RV_SQUARE_ROUND
	DVE_RV_BOTHSIDES

Edge Selection Based on Corners

Edge Selection Based on Corners selection function includes: *corner_edges()* (drc).

external()

Measures the distance between outside edges of polygons of different layers.

See also: *dve_drc()* (drc)

Syntax

dve_drc (external (inLayer1, inLayer2) operator distance [, msgString]
[, qualifierName, qualifierValue...]);

where:

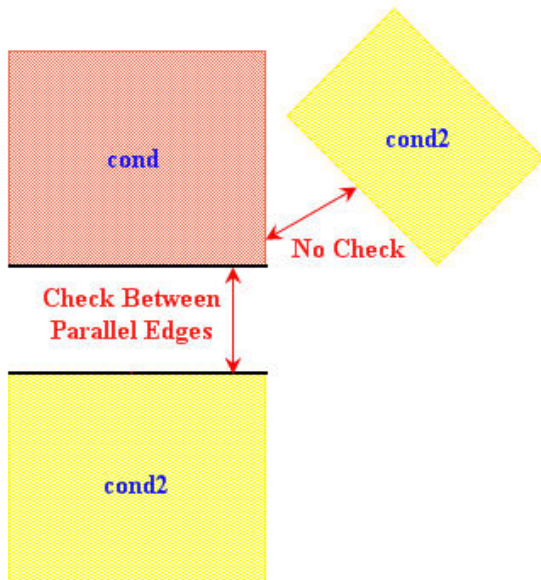
<i>inLayer1</i>	Containing polygon layer
<i>inLayer2</i>	Contained polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the selected error segments
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

Edge Qualifiers

DVE_RN_EDGE_ANGLES (drc)
DVE_RN_ANGLE_TOLERANCE (drc)
DVE_RN_SEPARATE (drc)
DVE_RN_TOUCH (drc)
DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO (drc)
DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM (drc)
DVE_RN_UPPER_BOUND (drc)

Example

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
drcError += dve_drc (external (lyrCond, lyrCond2) < 4.0,
    "Outside edges of metal layers < 4.0",
    DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL);
```



gap()

Measures the distance between outside edges of different polygons of the same layer.

See also: *dve_drc()* (drc)

Syntax

dve_drc (gap (inLayer) operator distance [, msgString] [,qualifierName, qualifierValue...]);

where:

<i>inLayer</i>	A polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the selected error segments
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

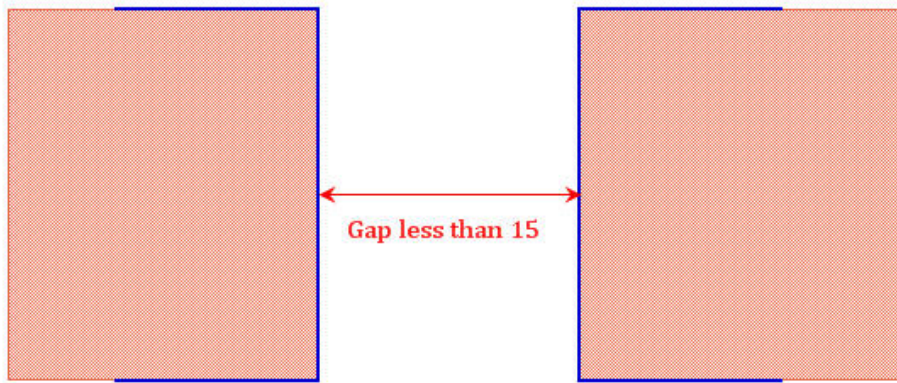
Edge Qualifiers

DVE_RN_EDGE_ANGLES (drc)
DVE_RN_ANGLE_TOLERANCE (drc)
DVE_RN_SEPARATE (drc)
DVE_RN_TOUCH (drc)
DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO (drc)
DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM (drc)
DVE_RN_UPPER_BOUND (drc)

Examples

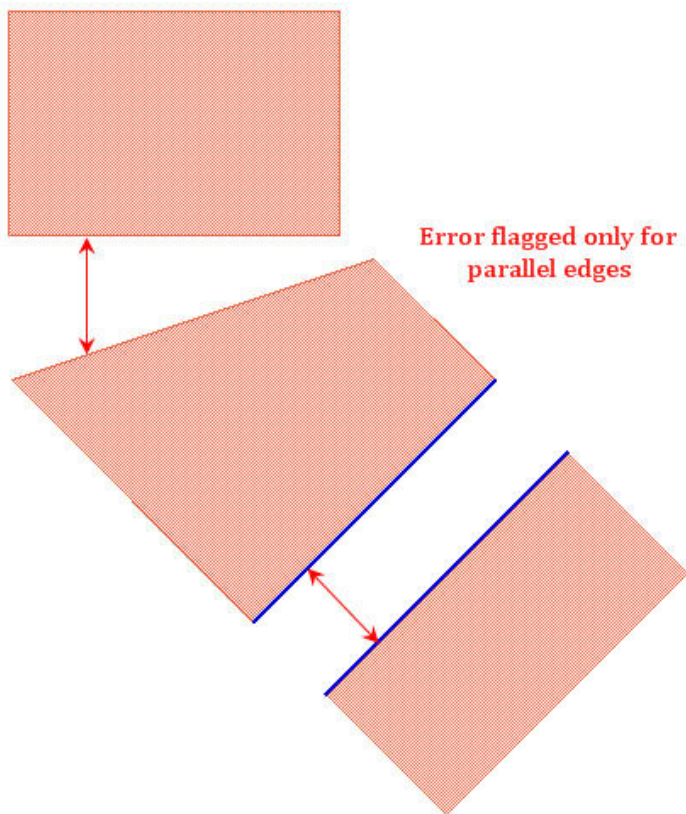
1. Checking gap between edges

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(gap(lyr_cond) < 15.00,
    "Gap of layer cond < 15.0");
```



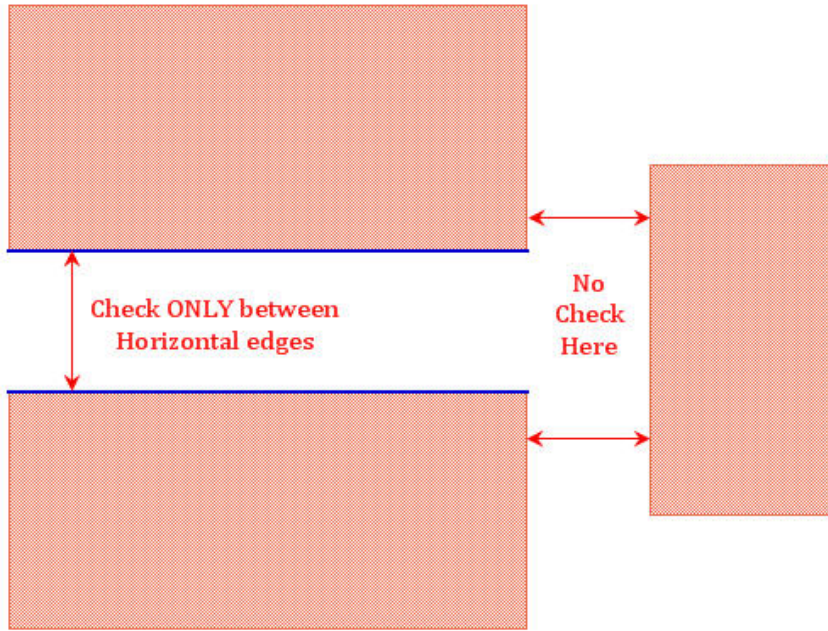
Checking between parallel edges

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(gap(lyr_cond) < 15.00,
    "Gap of layer cond < 15.0",
    DVE_RN_EDGE_ANGLES,DVE_RV_PARALLEL);
```



Checking between horizontally aligned edges

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(gap(lyr_cond) < 15.00,
    "Gap of layer cond < 15.0",
    DVE_RN_SLOPE,DVE_RV_HORIZONTAL);
```

i Likewise use a *slope qualifier* (drc) to activate a rule step only if it has a specified slope

internal()

Measures clearance from the inside of one edge of a polygon to the inside of another edge of a different polygon.

See also: *dve_drc()* (drc)

Syntax

dve_drc (internal (inLayer1, inLayer2) operator distance [, msgString] [, qualifierName, qualifierValue...]);

where:

<i>inLayer1</i>	Containing polygon layer
<i>inLayer2</i>	Contained polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the selected error segments
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

Edge Qualifiers

DVE_RN_EDGE_ANGLES (drc)

DVE_RN_ANGLE_TOLERANCE (drc)

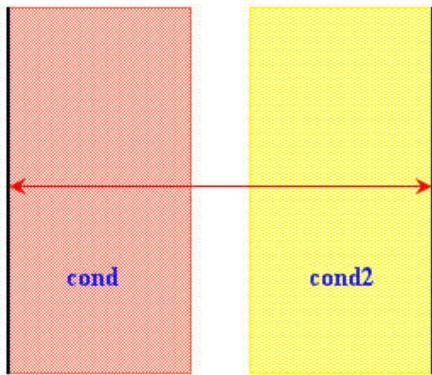
DVE_RN_SEPARATE (drc)*DVE_RN_TOUCH* (drc)*DVE_RN_SLOPE*, *DVE_RN_SLOPE_FROM*, *DVE_RN_SLOPE_TO* (drc)*DVE_RN_TEMPLATE*, *DVE_RN_TEMPLATE_TO*, *DVE_RN_TEMPLATE_FROM* (drc)*DVE_RN_UPPER_BOUND* (drc)

Example

```

decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
drcError += dve_drc (internal (lyrCond, lyrCond2) < 4.0,
    "Inside edges < 4.0");

```



nests()

Measures enclosure distance from the outside of the contained polygon to the inside of the containing polygon.

See also: *dve_drc()* (drc)

Syntax

```

dve_drc (nests (inLayer1, inLayer2) operator distance
[, msgString] [, qualifierName, qualifierValue...]);

```

where:

<i>inLayer1</i>	Contained polygon layer
<i>inLayer2</i>	Containing polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the selected error segments
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

Edge Qualifiers

DVE_RN_EDGE_ANGLES (drc)
DVE_RN_ANGLE_TOLERANCE (drc)
DVE_RN_SEPARATE (drc)
DVE_RN_TOUCH (drc)
DVE_RN_SLOPE, *DVE_RN_SLOPE_FROM*, *DVE_RN_SLOPE_TO* (drc)
DVE_RN_TEMPLATE, *DVE_RN_TEMPLATE_TO*, *DVE_RN_TEMPLATE_FROM* (drc)
DVE_RN_UPPER_BOUND (drc)

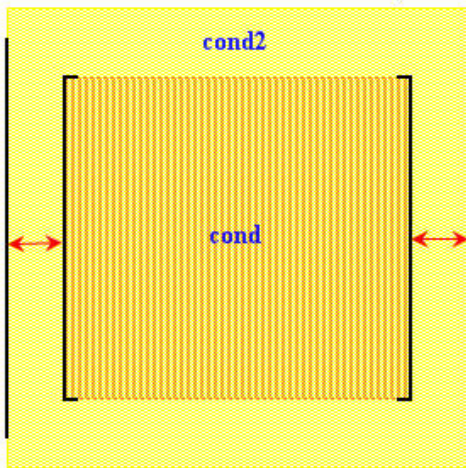
i The nests() command checks the similar edges that are checked by the *contains()* (drc) command. The only difference between them being the order of layers passed.

Examples

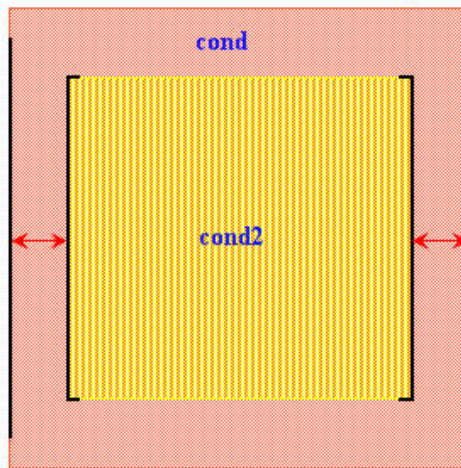
```

decl lyr_cond = dve_import_layer("cond");
decl lyr_cond2 = dve_import_layer("cond2");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc (nests (lyr_cond, lyr_cond2) < 3.0,
  "Case1 : Clearance from contained to containing layers < 3.0");
lyr_error += dve_drc (nests (lyr_cond2, lyr_cond) < 3.0,
  "Case2 : Clearance from contained to containing layers < 3.0");
  
```

Case1 : cond is the contained layer



Case2 : cond2 is the contained layer

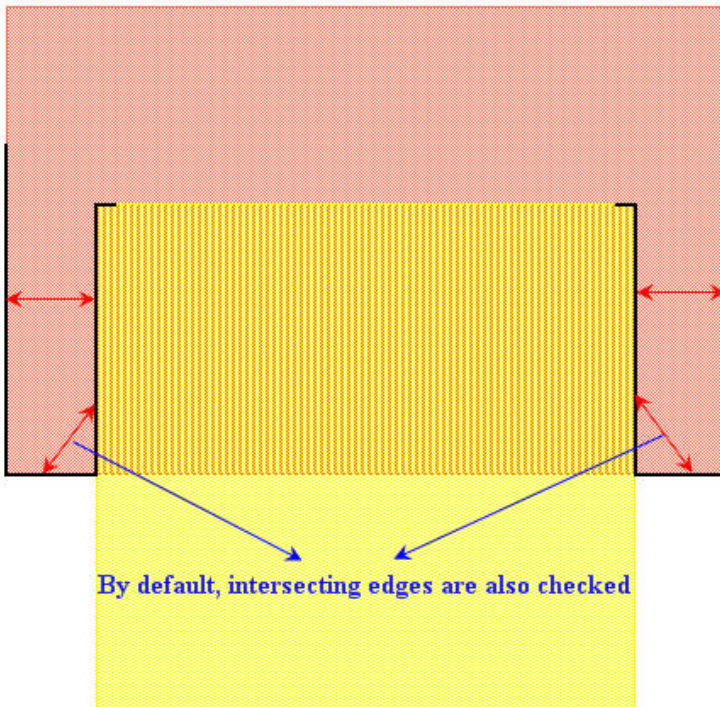


i The order of the layers is important while using the nests() command. First layer is the contained layer and second layer is the containing layer. By default, the intersecting edges are also checked. To ignore it, use the qualifier *DVE_RN_SEPARATE* (drc) with the resource value 'DVE_RV_SEPARATE'

Default checking,

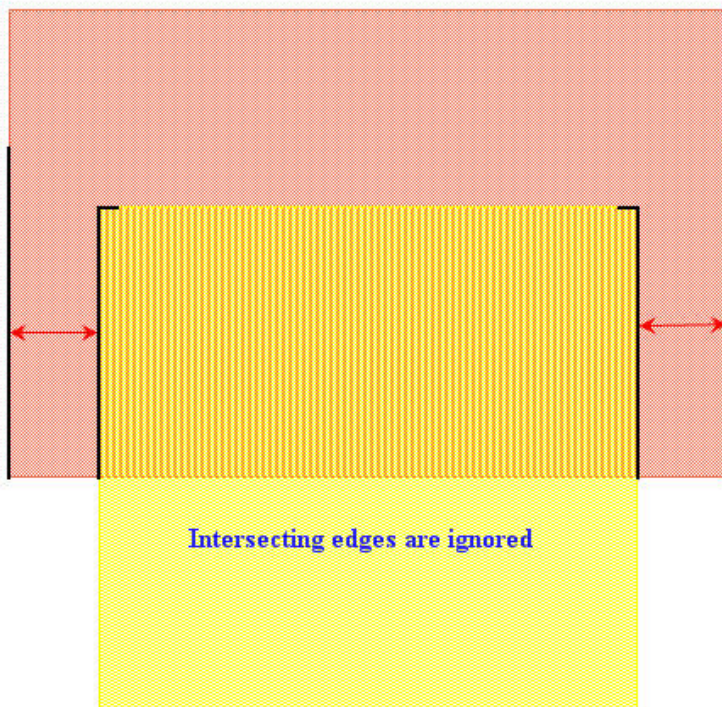
```

decl lyr_cond = dve_import_layer("cond");
decl lyr_cond2 = dve_import_layer("cond2");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc (nests (lyr_cond2, lyr_cond) < 3.0,
  "Clearance from contained to containing layers < 3.0",
);
  
```



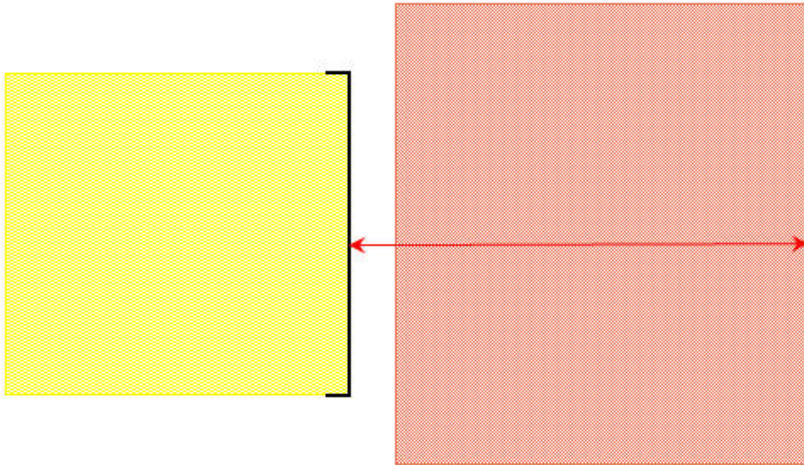
Using *DVE_RN_SEPARATE* (drc) qualifier,

```
decl lyr_cond = dve_import_layer("cond");  
decl lyr_cond2 = dve_import_layer("cond2");  
decl lyr_error = dve_export_layer(101);  
lyr_error += dve_drc (nests (lyr_cond2, lyr_cond) < 3.0,  
    "Clearance from contained to containing layers < 3.0",  
    DVE_RN_SEPARATE,DVE_RV_SEPARATE);
```



Note
 The nests() command will also check on the edges even when the two different layers' polygons are not included in each other if the clearance distance satisfies the constraint, which is, '**from Outside Edge of First Layer to Inside Edge of Second Layer**'. A user may see this behavior as a false error but actually the DRC engine is flagging the error when the constraint is satisfied. The DRC engine will not check which is a containing or a contained layer.

From Outside Edge of First layer to Inside Edge of Second layer



Tip
 To avoid a situation as shown above, declare a work layer which has a Boolean AND operation of the two layers in consideration. The code may be modified as:

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_cond2 = dve_import_layer("cond2");
decl lyr_error = dve_export_layer(101);
decl temp1 = dve_bool_and(lyr_cond2,lyr_cond);
lyr_error += dve_drc (nests (temp1, lyr_cond) < 3.0,
"Clearance from contained to containing layers < 3.0",
DVE_RN_SEPARATE,DVE_RV_SEPARATE);
```

notch()

Measures the distance between outside edges of the same polygon on the given layer.

See also: *dve_drc()* (drc)

Syntax

```
dve_drc (notch (inLayer) operator distance [, msgString]
[,qualifierName, qualifierValue...]);
```

where:

<i>inLayer</i>	A polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the selected error segments
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

Edge Qualifiers

DVE_RN_EDGE_ANGLES (drc)
DVE_RN_ANGLE_TOLERANCE (drc)
DVE_RN_SEPARATE (drc)
DVE_RN_TOUCH (drc)
DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO (drc)
DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM (drc)
DVE_RN_UPPER_BOUND (drc)

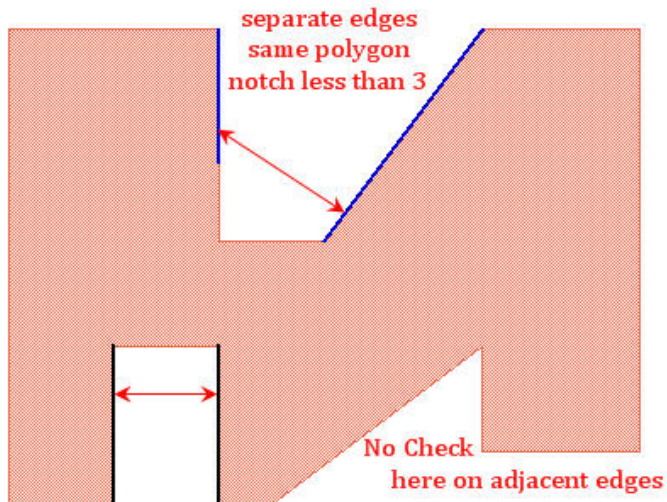
Examples


1. Checking *notch* between edges

```

decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(notch(lyr_cond) < 3.00,
  "Outside edges same polygon < 3.0");

```



 By default *notch()* will check on the separate edges

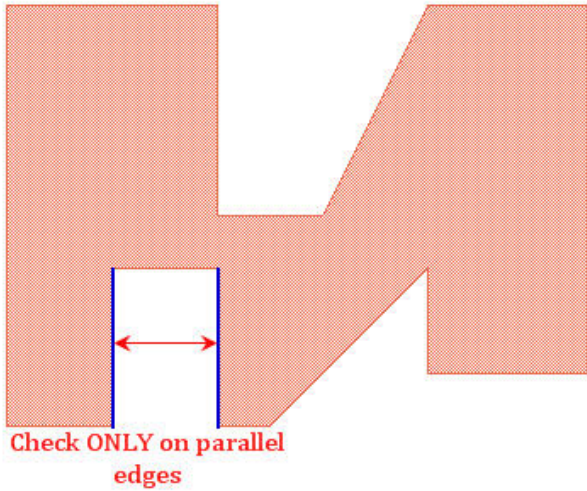
Checking *notch* between parallel edges

```

decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(notch(lyr_cond) < 3.00,

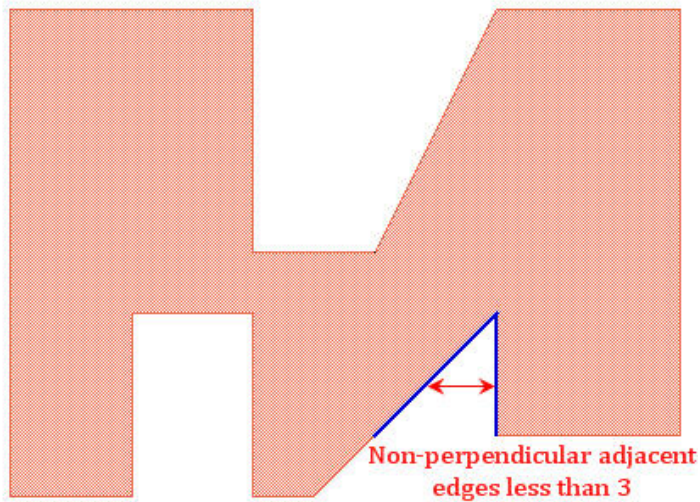
```

```
"Outside edges same polygon < 3.0",
DVE_RN_EDGE_ANGLES,DVE_RV_PARALLEL);
```



Checking *notch* between non perpendicular adjacent edges

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(notch(lyr_cond) < 3.00,
    "Outside edges same polygon < 3.0",
    DVE_RN_EDGE_ANGLES,DVE_RV_NOT_PERPENDICULAR,
    DVE_RN_SEPARATE,DVE_RV_NOT_SEPARATE);
```



off_grid()

Flags edges whose end points fall off a specified grid.

See also: *dve_drc()* (drc)

Syntax

dve_drc (off_grid (inLayer, grid) [,msgString]);

where:

<i>inLayer</i>	A polygon layer
<i>grid</i>	A specified grid
<i>msgString</i>	A string value that will be attached to the selected error segments

Example

```
decl lyrCond = dve_import_layer ("cond");
decl drcError = dve_export_layer ("ads_drc_error");
drcError += dve_drc (off_grid (lyrCond, 0.5),
    "Conductive metal is off grid");
```

Edge Compensation

Edge Compensation selection function includes: *compensate()* (drc), and *dve_segsize()* (drc).

single_clearance()

Measures the distance between edges of polygons on the same layer.

See also: *dve_drc()* (drc)

Syntax

dve_drc (single_clearance (inLayer) operator distance [, msgString] [,qualifierName, qualifierValue...]);

where:

<i>inLayer</i>	A polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the selected error segments
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

Edge Qualifiers

DVE_RN_POLARITY, DVE_RN_POLARITY_FROM, DVE_RN_POLARITY_TO (drc)

DVE_RN_STRUCTURE (drc)

DVE_RN_EDGE_ANGLES (drc)

DVE_RN_ANGLE_TOLERANCE (drc)

DVE_RN_SEPARATE (drc)

DVE_RN_TOUCH (drc)

DVE_RN_SLOPE, *DVE_RN_SLOPE_FROM*, *DVE_RN_SLOPE_TO* (drc)

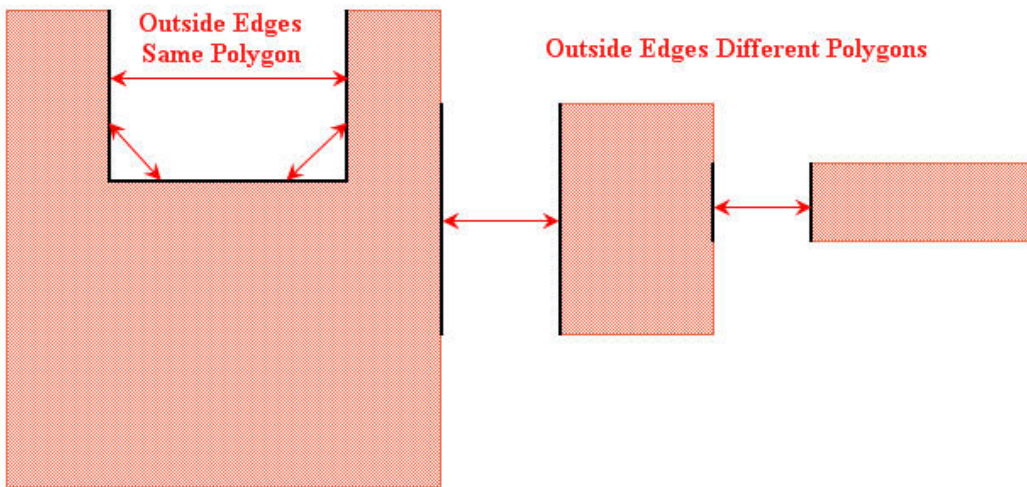
DVE_RN_TEMPLATE, *DVE_RN_TEMPLATE_TO*, *DVE_RN_TEMPLATE_FROM* (drc)


DVE_RN_UPPER_BOUND (drc)

Examples

1. Single clearance check without any *Edge Qualifier* (drc).

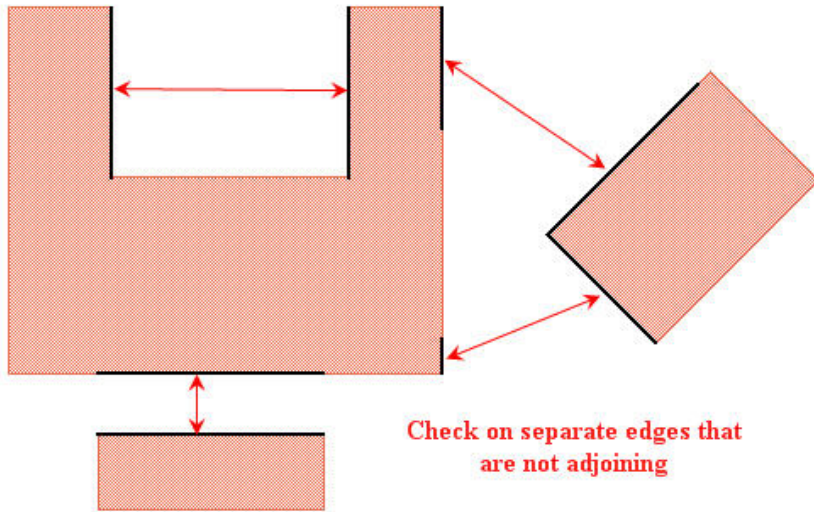
```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(single_clearance(lyr_cond) < 15.00,
    "Clearance of layer cond edges < 15.0");
```



 By default, `single_clearance()` will check on the outside edges of same and different polygons

Single clearance check on Non-Adjoining edges.

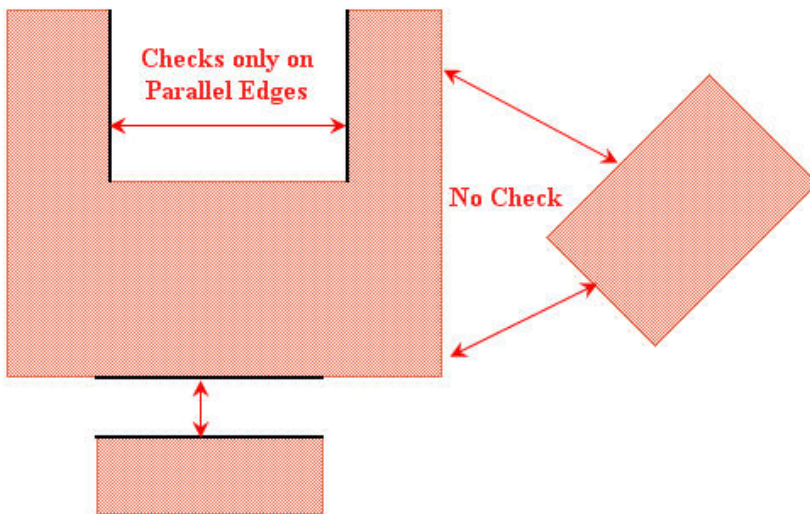
```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(single_clearance(lyr_cond) < 15.00,
    "Clearance of layer cond non-adjoining edges < 15.0",
    DVE_RN_SEPARATE, DVE_RV_SEPARATE);
```



Single clearance check on parallel edges.

```

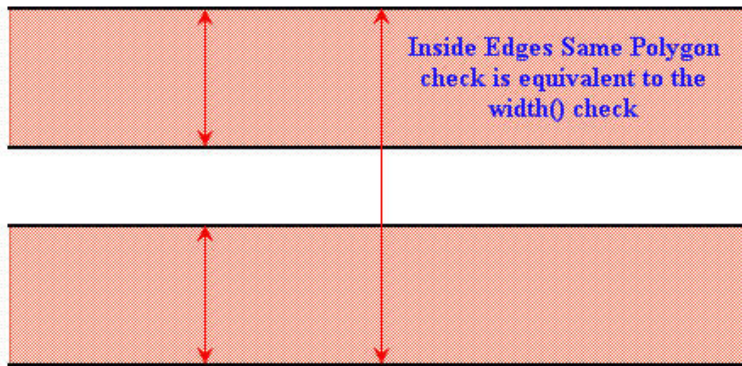
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(single_clearance(lyr_cond) < 15.00,
    "Clearance of layer cond parallel edges < 15.0",
    DVE_RN_EDGE_ANGLES,DVE_RV_PARALLEL);
    
```



Single clearance check on Inside edges of Same or Different Polygons that are parallel.

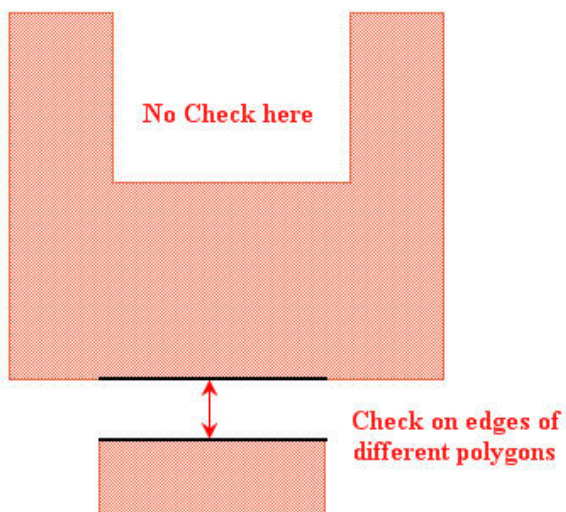
```

decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(single_clearance(lyr_cond) < 15.00,
    "Clearance of layer cond inside edges < 15.0",
    DVE_RN_EDGE_ANGLES,DVE_RV_PARALLEL,
    DVE_RN_POLARITY,DVE_RV_INSIDE);
    
```

Parallel Inside Edges of same and different polygons

Single clearance check between edges ONLY of different polygons.

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(single_clearance(lyr_cond) < 15.00,
    "Clearance of layer cond different polygon edges < 15.0",
    DVE_RN_STRUCTURE, DVE_RV_DIFF_POLYGON);
```

**spacing()**

Simultaneously measures the distance between outside edges of different polygons of the same layer (gap) and outside edges of the same polygon (notch).

See also: *dve_drc()* (drc)

Syntax

```
dve_drc (spacing (inLayer) operator distance [, msgString]
[, qualifierName, qualifierValue...]);
```

where:

<i>inLayer</i>	A polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the selected error segments
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

Edge Qualifiers

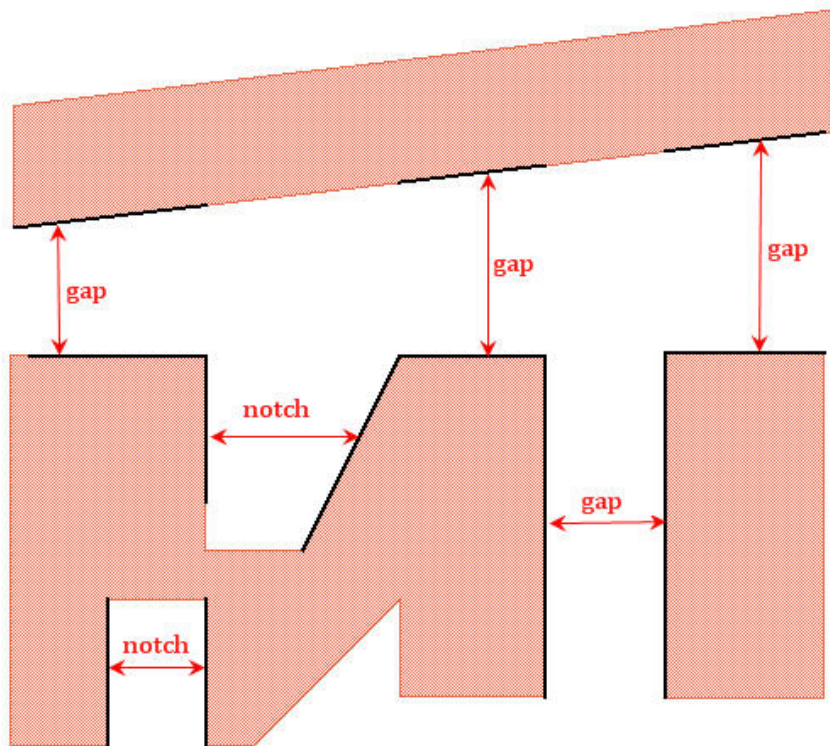
DVE_RN_EDGE_ANGLES (drc)
DVE_RN_ANGLE_TOLERANCE (drc)
DVE_RN_SEPARATE (drc)
DVE_RN_TOUCH (drc)
DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO (drc)
DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM (drc)
DVE_RN_UPPER_BOUND (drc)


Examples

1. Spacing check will include the *notch* and the *gap* checks.

```

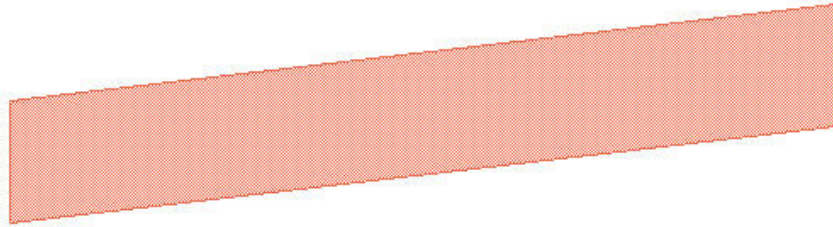
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(spacing(lyr_cond) < 3.00,
  "Outside edges of same and different polygons < 3.0");
  
```



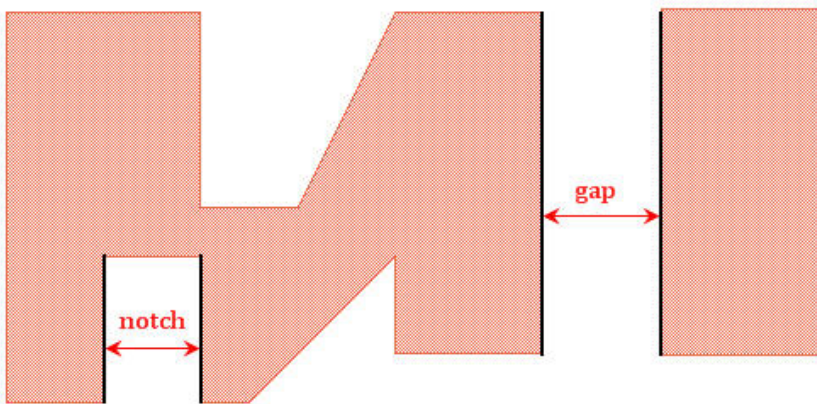
 By default spacing() will check on the separate edges

Spacing check between parallel edges.

```
decl lyr_cond = dve_import_layer("cond");  
decl lyr_error = dve_export_layer(101);  
lyr_error += dve_drc(spacing(lyr_cond) < 3.00,  
    "Outside edges of same and different polygons < 3.0",  
    DVE_RN_EDGE_ANGLES,DVE_RV_PARALLEL);
```

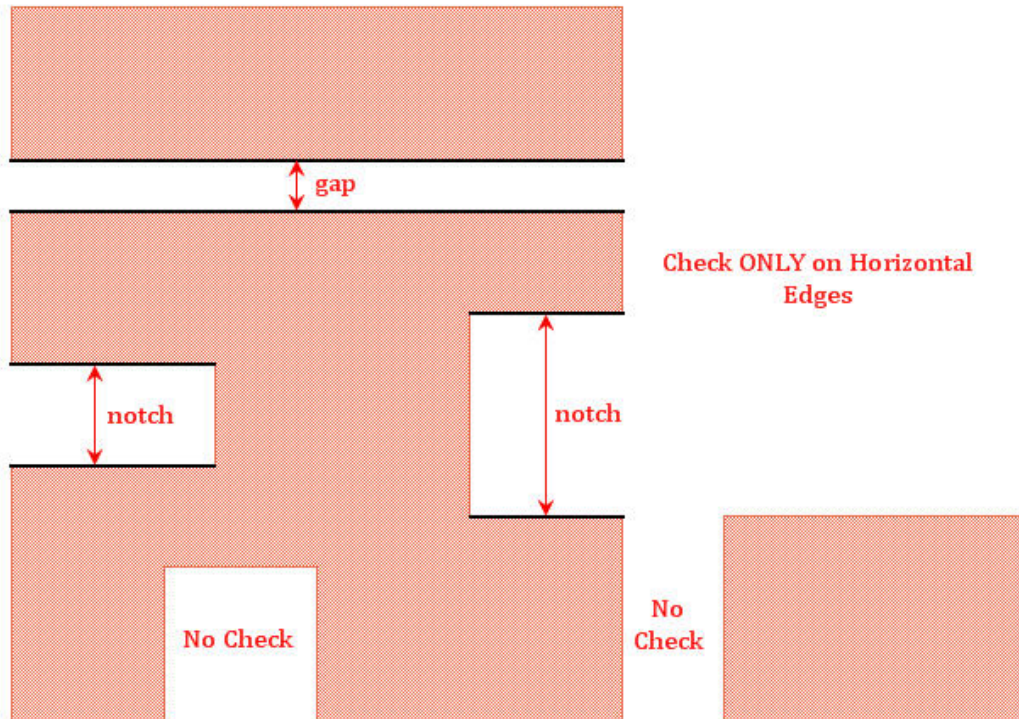


Check ONLY on Parallel Edges



Spacing check between horizontally aligned edges.

```
decl lyr_cond = dve_import_layer("cond");  
decl lyr_error = dve_export_layer(101);  
lyr_error += dve_drc(spacing(lyr_cond) < 3.00,  
    "Outside edges of same and different polygons < 3.0",  
    DVE_RN_SLOPE,DVE_RV_HORIZONTAL);
```



i Likewise use a *slope qualifier* (*drc*) to activate a rule step only if it has a specified slope

width()

A DRC clearance function to check from the inside of one edge of a polygon to the inside of another edge of the same polygon.

See also: *dve_drc()* (*drc*)

Syntax

dve_drc (*width* (*inLayer*) *operator* *distance* [, *msgString*] [, *qualifierName*, *qualifierValue*, ...]);

where:

<i>inLayer</i>	A polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to n
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the selected error segments
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

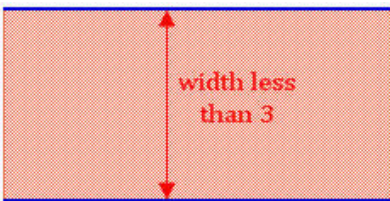
Edge Qualifiers

DVE_RN_EDGE_ANGLES (drc)
DVE_RN_ANGLE_TOLERANCE (drc)
DVE_RN_SEPARATE (drc)
DVE_RN_TOUCH (drc)
DVE_RN_SLOPE, *DVE_RN_SLOPE_FROM*, *DVE_RN_SLOPE_TO* (drc)
DVE_RN_TEMPLATE, *DVE_RN_TEMPLATE_TO*, *DVE_RN_TEMPLATE_FROM* (drc)
DVE_RN_UPPER_BOUND (drc)

Example

1. Checking width of layer

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(width(lyr_cond) < 3.00,
    "Width of layer cond < 3.0");
```



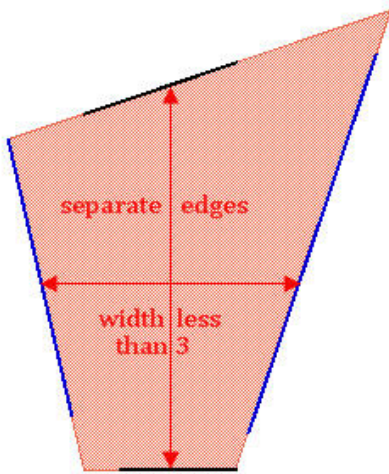
2. Checking on parallel edges

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(width(lyr_cond) < 3.00,
    "Width of layer cond < 3.0",
    DVE_RN_EDGE_ANGLES,DVE_RV_PARALLEL);
```



3. Checking on separate edges

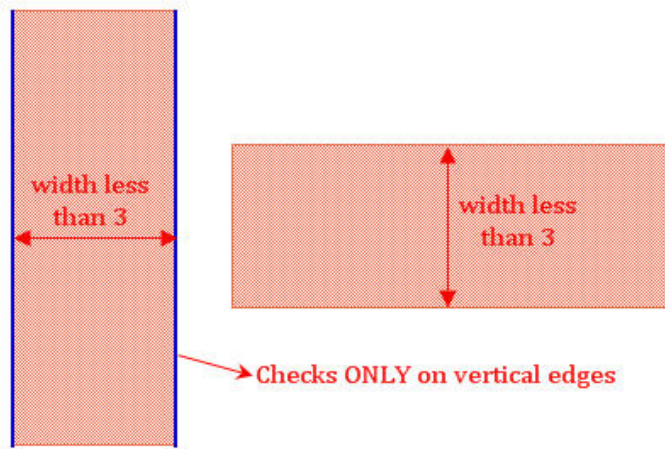
```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(width(lyr_cond) < 3.00,
    "Width of layer cond < 3.0",
    DVE_RN_SEPARATE,DVE_RV_SEPARATE);
```



i By default width() will check on the separate edges

4. Checking on vertically aligned edges

```
decl lyr_cond = dve_import_layer("cond");
decl lyr_error = dve_export_layer(101);
lyr_error += dve_drc(width(lyr_cond) < 3.00,
    "Width of layer cond < 3.0",
    DVE_RN_SLOPE,DVE_RV_VERTICAL);
```



i Likewise use a *slope qualifier* (drc) to activate a rule step only if it has a specified slope

DRC Layer Management Commands

This section describes the DRC Layer Management commands used to import and export layers.

- *dve_import_text_layer()* (drc)
- *dve_export_layer()* (drc)
- *dve_identify_cell_layer()* (drc)
- *dve_import_cell_layer()* (drc)
- *dve_import_layer()* (drc)

dve_import_text_layer()

This command is useful for selecting polygons based on whether they contain certain text. It copies the layer data of texts into an import layer that can be used in a rule command and returns an import layer.

See also: *dve_export_layer()* (drc), *dve_import_layer()* (drc)

Syntax

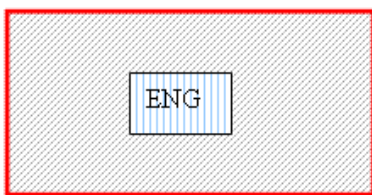
```
inputLayer = dve_import_text_layer (textstring);
```

where:

textstring is a string literal that is present in the design in the form of ADS Text

Example

```
decl lyrText = dve_import_text_layer("ENG");
decl lyrGate = dve_import_layer("t_Gate");
decl lyrErr101 = dve_export_layer(101);
decl lyrPoly = dve_drc(poly_inter_layer(lyrGate,
lyrText),DVE_RN_INTER_SELECT,DVE_RV_ACCEPT,DVE_RN_INTER_CODE, DVE_RV_ENCLOSE);
lyrErr101 += dve_drc(all_edges(lyrPoly),"Excluded due to ENG marker");
```



← Edges of t_Gate is highlighted as the bounding box of "ENG" is contained completely inside t_Gate



← Since "ENG" is not completely inside t_Gate, it is not highlighted as per given the rules written above

Note

The string literal matching is exact and case sensitive. For example, if the design contains text as "ENGLISH", it will be not be considered/included in the *dve_import_text_layer* rule operation.

dve_export_layer()

Used to export DRC error information. Data written to an export layer will be directly exported back to the layout editor. Returns an export layer.

See also: *dve_import_layer()* (drc), *dve import cell layer()* (drc)

Syntax

```
exportLayer = dve_export_layer (layerId);
```

where:

<i>layerId</i>	is the string layer name or integer layer number of an existing design layer
----------------	--

Example

```
// Import layers
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
// Export layers
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrError102 = dve_export_layer ("error102");
// Work layer
decl lyrOverlap = NULL;
// Export DRC error directly to an export layer
drcError += dve_drc (width (lyrCond) < 4.0, "Metal width less than 4.0");
lyrOverlap = dve_bool_and (lyrCond, lyrCond2);
lyrError102 += dve_drc (all_edges (lyrOverlap), "Metal layers overlap");
```

dve_identify_cell_layer()

Processes the design layers prior to input into the DRC engine. Uses a AEL callback function for doing the customization in identification steps. Returns reference to the layer returned by the callback function. If nothing is returned from the callback function, a rules-compilation error is displayed.

See also: *dve_export_layer()* (drc), *dve_import_layer()* (drc)

Syntax

```
decl drc_layA = dve_identify_cell_layer("DEVICE_TYPE1", "layA", "DEV1_Identify_cb");
```

where:

DEVICE_TYPE1	device name that needs to be identified. A string value.
layA	layer name or layer number. A string or an integer value.
DEV1_Identify_cb	Callback to identify the device. A string value.

Example 1

Callback definition AEL file:

```
defun identify_cond_layer_of_dev1_cb(dev1, layerId)
{
  de_add_layer("drc_DevRec",100,100,100,1,10,0,2,0,1,"*",1);
  // use any AEL layer manipulation routines to manipulate drc_DevRec layer
  return 100; // the drc_DevRec layer
}
```

DRC Rules file:

```
decl drc_cond = dve_identify_cell_layer("DEV1", "cond", "identify_cond_layer_of_dev1_cb");
decl drc_error = dve_export_layer(101);
drc_error += dve_drc(poly_area(drc_cond) > 100.0);
```

dve_import_cell_layer()

Used to get design data from the layout editor into the design verification process. Copies instance artwork from the layout editor onto an import layer that can be used in a rule command. Returns an import layer.

Note
dve_import_cell_layer can not be used on components with a sub-circuit.

See also: *dve_export_layer()* (drc), *dve_import_layer()* (drc)

Syntax

```
inputLayer = dve_import_cell_layer (layerId, cellName [, callBack]);
```

where:

layerId	is the string layer name or integer layer number of an existing design layer
cellName	is the name of the component or "*"
callBack	is the name of the component selection callback

Example 1

```
decl lyrCond = dve_import_cell_layer ("cond", "MLIN");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = dve_bool_and (lyrCond, lyrCond2);
drcError += dve_drc (all_edges (lyrPoly), "MLIN overlapping cond2");
```

Example 2

```
decl lyrCond = dve_import_cell_layer ("cond", "*");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = dve_bool_and (lyrCond, lyrCond2);
drcError += dve_drc (all_edges (lyrPoly), "Component cond metal overlapping cond2");
```

Example 3

```

defun myprocess_MLIN_cb(instH, layerId, cellName)
{
// Add selection criteria
// Return true if this instance of component MLIN should be included in the cell layer
return TRUE;
}
decl lyrCond = dve_import_cell_layer ("cond", "MLIN", "myprocess_MLIN_cb");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = dve_bool_and (lyrCond, lyrCond2);
drcError += dve_drc (all_edges (lyrPoly), "MLIN overlapping cond2");

```

dve_import_layer()

Used to get design data from the layout editor into the design verification process. Copies layer data from the layout editor onto an import layer that can be used in a rule command. Returns an import layer.

See also: *dve_export_layer()* (drc), *dve import cell layer()* (drc)

Syntax

```
inputLayer = dve_import_layer (layerId[,purpose]);
```

where:

layerId	is the string layer name or integer layer number of an existing design layer
purpose	is an optional string or integer argument specifying the purpose of the above layerId

If *purpose* is not specified, the DRC will pick up shapes of all purposes belonging to *layerId*. The second argument thus provides an additional level of filtering of shapes belonging to a particular layerId.

Example

```

decl lyrCond = dve_import_layer ("cond");
decl lyrResiFill = dve_import_layer("resi","fill");
decl drcError = dve_export_layer ("ads_drc_error");
drcError += dve_drc (width (lyrCond) < 4.0, "Metal width less than 4.0");
drcError += dve_drc (width (lyrResiFill) < 6.0, "Resi:Fill width less than 6.0");

```

Macros

This section describes macros. Two macros that are currently defined are:

- *intrusion()* (drc)
- *protrusion()* (drc)

intrusion()

Checks the intrusion of BOTTOM layer into TOP layer (see *Convention for TOP and BOTTOM layers* (drc)).

Syntax

```
errorLayer = dve_drc (intrusion (inLayer1, inLayer2) operator distance [,msgString]);
```

where:

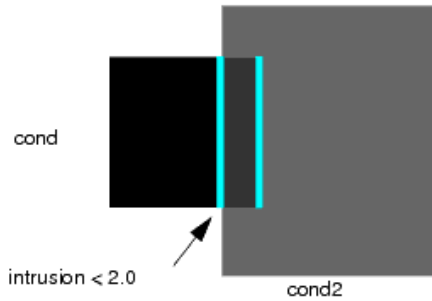
<i>errorLayer</i>	Layer with error segments
<i>inLayer1</i>	Polygon TOP layer
<i>inLayer2</i>	Polygon BOTTOM layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the error segments

Method

```
decl mergeGate = dve_bool_and(inLayer2, inLayer1);
decl selectGate = dve_drc (poly_edge_code (mergeGate),DVE_RN_EDGE_SELECT,
DVE_RV_ACCEPT_ANY, DVE_RN_PATH_CODE, DVE_RV_BIT);
decl overlapGate = dve_bool_and (selectGate, inLayer2);
decl notGate = dve_bool_not (inLayer1, overlapGate);
errorLayer = dve_drc(double_clearance(overlapGate, notGate)
    @operator @distance, @msgString,
    DVE_RN_POLARITY_FROM, DVE_RV_INSIDE,
    DVE_RN_POLARITY_TO, DVE_RV_OUTSIDE,
    DVE_RN_TEMPLATE, DVE_RV_OPPOSITE,
    DVE_RN_SEPARATE, DVE_RV_SEPARATE);
```

Example

```
decl lyrCond = dve_import_layer("cond");
decl lyrCond2 = dve_import_layer("cond2");
decl drcError = dve_export_layer("ads_drc_error");
drcError = dve_drc (intrusion (lyrCond2,lyrCond) <= 2.0);
```



protrusion()

Checks the extension of TOP layer out of BOTTOM layer (see *Convention for TOP and BOTTOM layers (drc)*).

Syntax

```
errorLayer = dve_drc (protrusion (inLayer1, inLayer2) operator distance [,msgString]);
```

where:

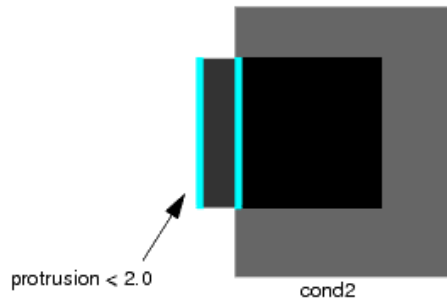
<i>errorLayer</i>	Layer with error segments
<i>inLayer1</i>	Polygon TOP layer
<i>inLayer2</i>	Polygon BOTTOM layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A distance value in layout units
<i>msgString</i>	A string value that will be attached to the error segments

Method

```
decl gatePoly = dve_drc (poly_inter_layer (inLayer1, inLayer2),
DVE_RN_INTER_CODE, DVE_RV_CUT);
decl proGatePoly = dve_bool_not (gatePoly, inLayer2);
errorLayer += dve_drc (double_clearance (proGatePoly, inLayer2)
@operator @distance,@msgString,
DVE_RN_POLARITY_FROM, DVE_RV_INSIDE,
DVE_RN_POLARITY_TO, DVE_RV_OUTSIDE,
DVE_RN_TEMPLATE, DVE_RV_OPPOSITE,
DVE_RN_SEPARATE, DVE_RV_SEPARATE);
```

Example

```
decl lyrCond = dve_import_layer("cond");
decl lyrCond2 = dve_import_layer("cond2");
decl drcError = dve_export_layer("ads_drc_error");
drcError = dve_drc (protrusion (lyrCond, lyrCond2) <= 2);
```



Merge Operations on Polygons

This section describes boolean operations on polygons:

- *dve_bool_and()* (drc)
- *dve_bool_not()* (drc)
- *dve_bool_or()* (drc)

and the commands for combining layers.

- *dve_merge()* (drc)
- *dve_self()* (drc)
- *dve_self_merge()* (drc)

dve_bool_and()

Merges overlapping polygons on two given layers. Returns: A polygon layer.

Syntax

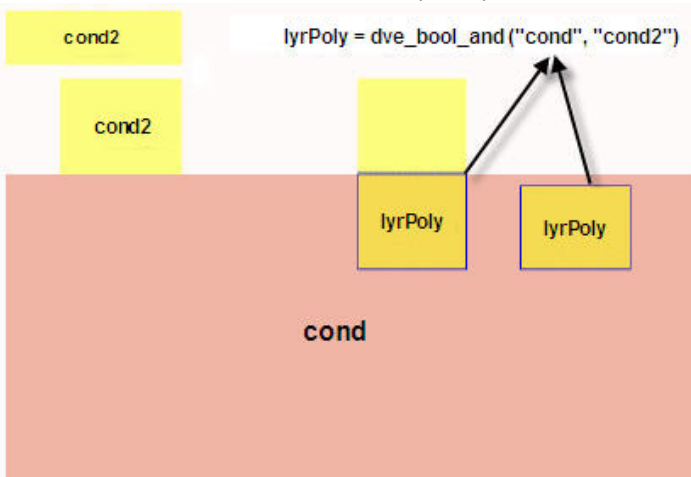
```
dve_bool_and (inLayer1, inLayer2);
```

where:

<i>inLayer1</i> , <i>InLayer2</i>	A polygon layer
-----------------------------------	-----------------

Example

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = NULL;
lyrPoly = dve_bool_and (lyrCond, lyrCond2);
drcError += dve_drc (all_edges (lyrPoly), "Conductive metal overlapping");
```



dve_bool_not()

Subtracts shapes in the second layer from shapes in the first layer. Returns: A polygon layer.

Syntax

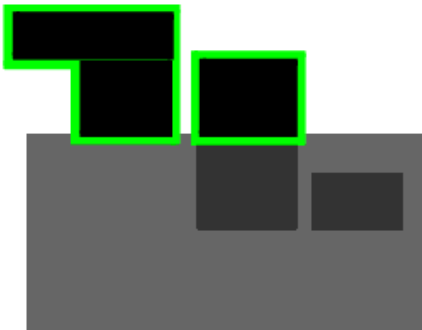
```
dve_bool_not (inLayer1, inLayer2);
```

where:

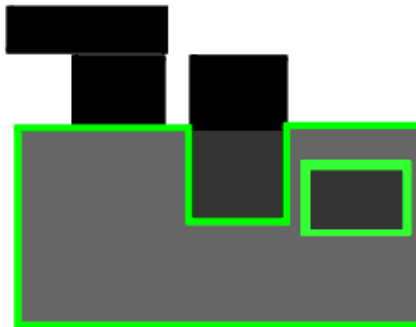
<i>inLayer1</i>	<i>InLayer2</i>	A polygon layer
-----------------	-----------------	-----------------

Example

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = NULL;
lyrPoly = dve_bool_not (lyrCond, lyrCond2);
drcError += dve_drc (all_edges (lyrPoly),
    "Conductive metal not overlapping");
lyrPoly = dve_bool_not (lyrCond2, lyrCond);
drcError += dve_drc (all_edges (lyrPoly),
    "Conductive metal not overlapping");
```



dve_bool_not(lyrCond, lyrCond2)



dve_bool_not(lyrCond2, lyrCond)

Notice that in the second example the polygon 4 on the layer 'cond' creates a hole.

dve_bool_or()

Merges overlapping shapes on a given layer. Returns: A polygon layer.

Syntax

```
outLayer = dve_bool_or (inLayer1 [, inLayer2]);
```

where:

<i>inLayer1, InLayer2</i>	A polygon layer
---------------------------	-----------------

Example

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = NULL;
lyrPoly = dve_bool_or (lyrCond, lyrCond2);
drcError += dve_drc (width (lyrPoly) < 3.0,
    "Conductive metal less than 3.0");
```



dve_merge()

Collects layers into one layer without modifying the shapes. Results of a combine operation can be used for performing merges, boolean operations and sizing operations. It is important to note that no merge or boolean operations are performed in the process. Returns: a polygon layer.

Syntax

```
dve_merge ( inLayer1 [, inLayer2, . . . , inLayerN])
```

where:

<i>inLayer1, InLayer2, inLayerN</i>	A polygon layer that is not the result of a dve_merge
-------------------------------------	---

Example

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrDiel = dve_import_layer ("diel");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrMerge = NULL;
lyrMerge = dve_merge (lyrCond, lyrCond2, lyrDiel);
drcError += dve_drc (corner_edges (lyrMerge, 1.5, 90.5, 360.0),
    "Concave corner edges");
```

dve_self()

Merge shapes to eliminate overlaps. Returns: a polygon layer.

Syntax

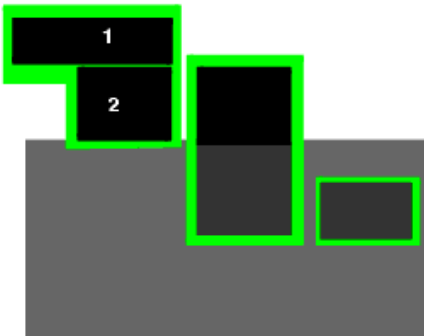
dve_self(inLayer)

where:

<i>inLayer1, InLayer2</i>	A polygon layer
---------------------------	-----------------

Example

```
decl lyrCond = dve_import_layer("cond");
decl drcError = dve_export_layer("ads_drc_error");
decl lyrCondMerged = NULL;
decl lyrCondMerged = dve_self(lyrCond);
drcError += dve_drc(all_edges(lyrCondMerged));
```



Note	Polygons 1 and 2 are merged.
-------------	------------------------------

dve_self_merge()

Merge shapes on the first specified layer and move them to the TOP level, collect shapes on the second specified layer and move them to the BOTTOM level (see *Convention for TOP and BOTTOM layers (drc)*). This command is particularly useful when used with polygon selection commands (see *Polygon Selection Based on Merge Properties (drc)*). Returns : A polygon layer.

Syntax

dve_self_merge (inLayer1, inLayer2)

where:

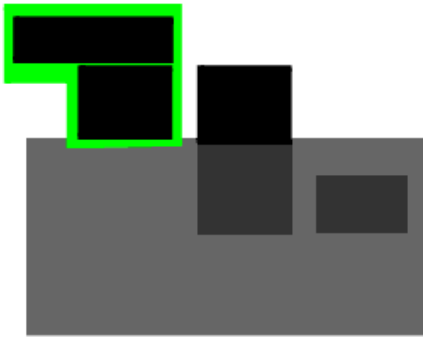
<i>inLayer1, InLayer2</i>	A polygon layer
---------------------------	-----------------

Example

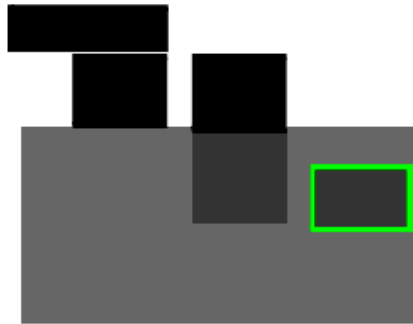
```

decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPolyMerge = NULL;
decl lyrPoly1 = NULL;
lyrPolyMerge = dve_self_merge (lyrCond2, lyrCond);
lyrPoly1 = dve_drc (poly_edge_code (lyrPolyMerge),
    DVE_RN_EDGE_SELECT, DVE_RV_ACCEPT_ALL,
    DVE_RN_PATH_CODE, DVE_RV_TOP);
drcError += dve_drc (all_edges (lyrPoly1),
    "Conductive metal outside");
lyrPoly1 = dve_drc (poly_edge_code (lyrPolyMerge),
    DVE_RN_EDGE_SELECT, DVE_RV_ACCEPT_ALL,
    DVE_RN_PATH_CODE, DVE_RV_TIB);
drcError += dve_drc (all_edges (lyrPoly1),
    "Conductive metal inside");

```



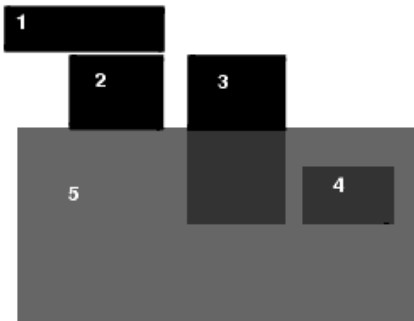
conductive metal outside



conductive metal inside

Example for Performing Boolean Operations

Boolean operations will be illustrated by the following example. Several polygons are placed on layer 'cond' (1,2,3,4). One polygon is placed on layer 'cond2' (5).



Note

Polygon1 butts against polygon 2 on the same layer, and polygon 2 butts against the outside of polygon 5 on a different layer.

Operations for Polygon Extraction from Edges

This section describes the DRC command used for polyextraction from edges. These functions include:

- *dve_plgout()* (drc)
- *dve_quadout()* (drc)

dve_plgout()

Extracts entire polygons from selected edges. If any section of a polygon is in error, then the entire polygon is extracted. Returns: A polygon layer.

See also: *dve_quadout()* (drc)

Syntax

dve_plgout (edgeLayer);
where:

<i>edgeLayer</i>	A layer containing selected edge segments. Edge segments are selected using the <i>dve_drc</i> command
------------------	--

Example

```
decl lyrCond = dve_import_layer ("cond");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrEdges1 = NULL;
decl lyrEdges2 = NULL;
decl lyrEdges3 = NULL;
decl lyrPolyInterconnect = NULL;
//Identify sections of interconnect metal w/width >=2.0 and width <= 3.0
lyrEdges1 = dve_drc (width (lyrCond) < 2.0);
lyrEdges2 = dve_drc (invert_edges (lyrEdges1));
lyrEdges3 = dve_drc (width (lyrEdges2) < 3.0);
lyrPolyInterconnect = dve_plgout (lyrEdges3);
drcError += dve_drc (all_edges (lyrPolyInterconnect),
    "Valid interconnect");
```

dve_quadout()

Extracts a quadrilateral from the selected error segments on the given layer. Returns: A polygon layer.

See also: *dve_plgout()* (drc)

Syntax

dve_quadout (edgeLayer);

where:

edgeLayer	A layer containing selected edge segments. Edge segments are selected using the dve_drc command
-----------	---

Example 1

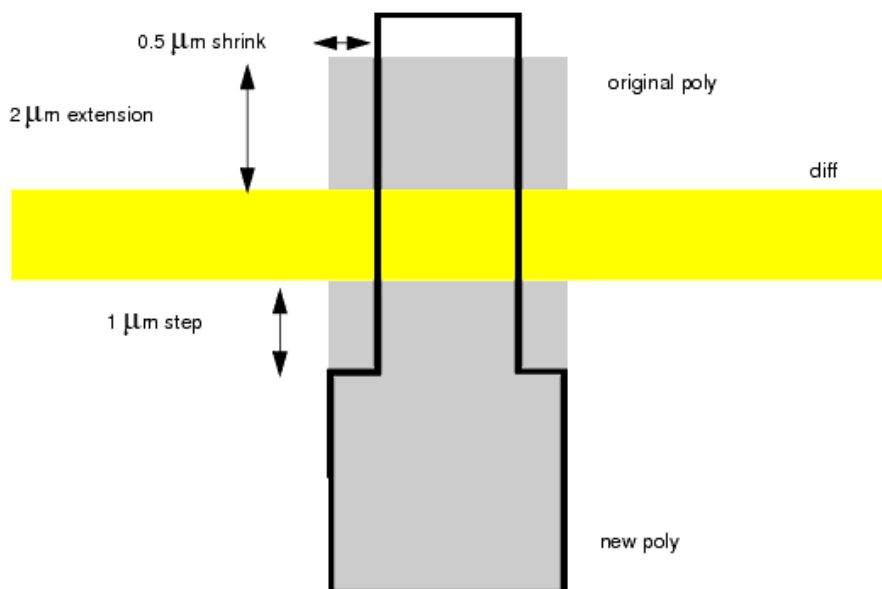
```
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrEdges = NULL;
decl lyrPoly = NULL;
decl lyrPolySmall = NULL;
lyrEdges = dve_drc (width (lyrCond2) < 3.0,
    DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL,
    DVE_RN_TEMPLATE, DVE_RV_OPPOSITE);
lyrPoly = dve_quadout (lyrEdges);
lyrPolySmall = dve_drc (poly_line_length (lyrPoly) < 4.0,
    DVE_RN_LINE_LENGTH, DVE_RV_MAX_LINE);
drcError += dve_drc (all_edges (lyrPolySmall),
    "Conductive metal length less than 4.0");
```

Example 2 - Geometric Compensation

In general, there are three main tools for doing geometric compensation that can be used in isolation or in combination:

- Use the size/undersize operators to play geometric tricks
- Use a DRC clearance rule with the quadout option to extract the space between two edges as a polygon
- Use any DRC rules to develop error segments, and then use the compensate command to add to or subtract from the polygon.

The following MOS problem demonstrates the use of the size/undersize and quadout operations.



The rules specify the following actions:

- Compensate the channel by pulling in the source/drain by 0.5 μm .
- If there is a small lead-away of poly (as shown at the top of the figure), maintain a 2 μm extension at the new shrunk length.
- If there is a big lead-away of poly (as shown at the bottom of the figure), step the poly 1 μm back from the side of the channel.

This involves a multi-step sizing operation:

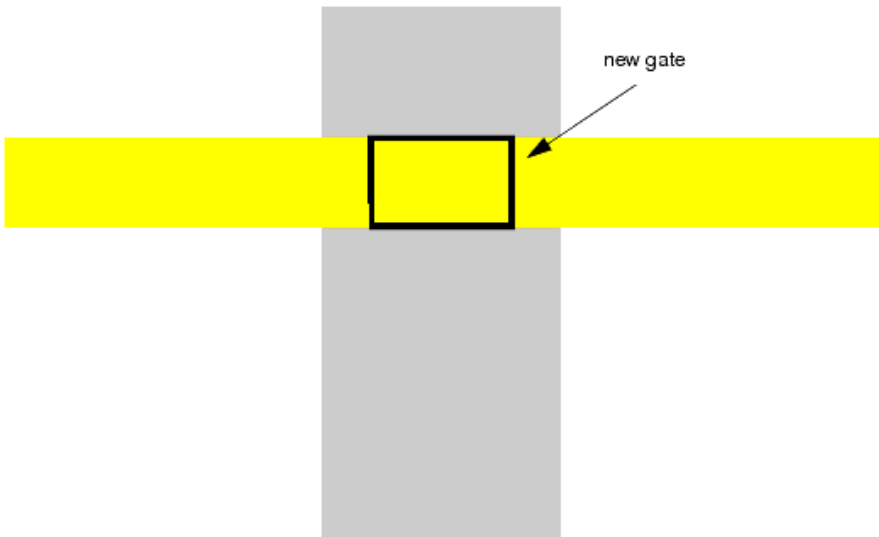
```
decl lyrPoly = dve_import_layer("poly");
decl lyrDiff = dve_import_layer("diff");
decl drcError = dve_export_layer("ads_drc_error");
decl lyrError102 = dve_export_layer("error102");
decl lyrError103 = dve_export_layer("error103");
decl lyrError104 = dve_export_layer("error104");
decl lyrShrunkPoly = NULL;
decl lyrNewGate = NULL;
decl lyrOldGate = NULL;
decl lyrOversizedOldGate = NULL;
decl lyrBigDiff = NULL;
decl lyrPolyExtension = NULL;
decl lyrBigPolyExtension = NULL;
decl lyrNestEdges = NULL;
decl lyrPolyFiller = NULL;
decl lyrNewPoly = NULL;
```

Start by undersizing the original polygon...

```
lyrShrunkPoly = dve_undersize(lyrPoly, 0.5);
```

... to produce a new gate polygon:

```
lyrNewGate = dve_bool_and(lyrShrunkPoly, lyrDiff);
drcError +=dve_drc(all_edges(lyrNewGate), "new gate");
```



Next, oversize diff by 1 micron. This produces a region that will be used in a later operation to cut away the step for the big lead-away.

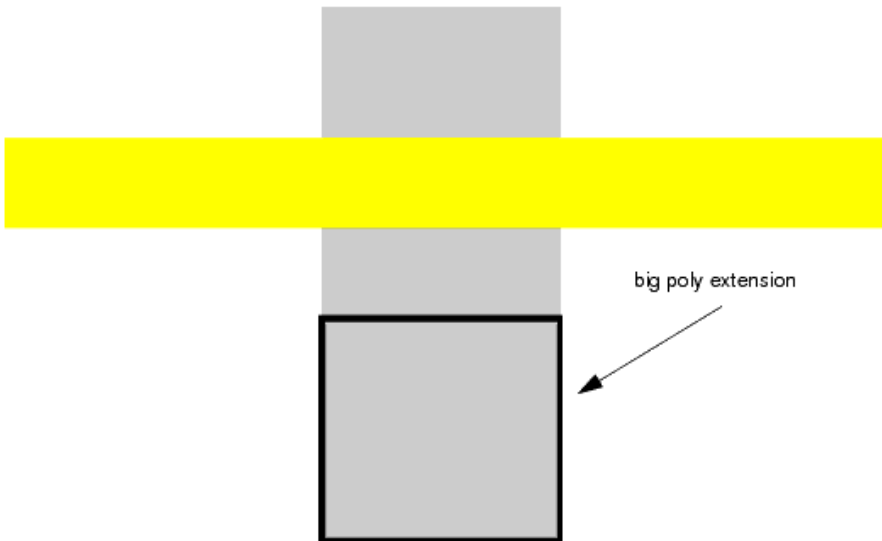
```
lyrBigDiff = dve_oversize(lyrDiff , 1.0);
```

Retain the extension of poly that is outside of the oversized diff zone

```
lyrPolyExtension = dve_bool_not(lyrPoly, lyrBigDiff);
```

But only the big extensions are required. Determine these zones using the area of the extensions:

```
lyrBigPolyExtension = dve_drc(poly_area(lyrPolyExtension ) > 3.0);
lyrError102 +=dve_drc(all_edges(lyrBigPolyExtension ),
"lyrBigPolyExtension");
```

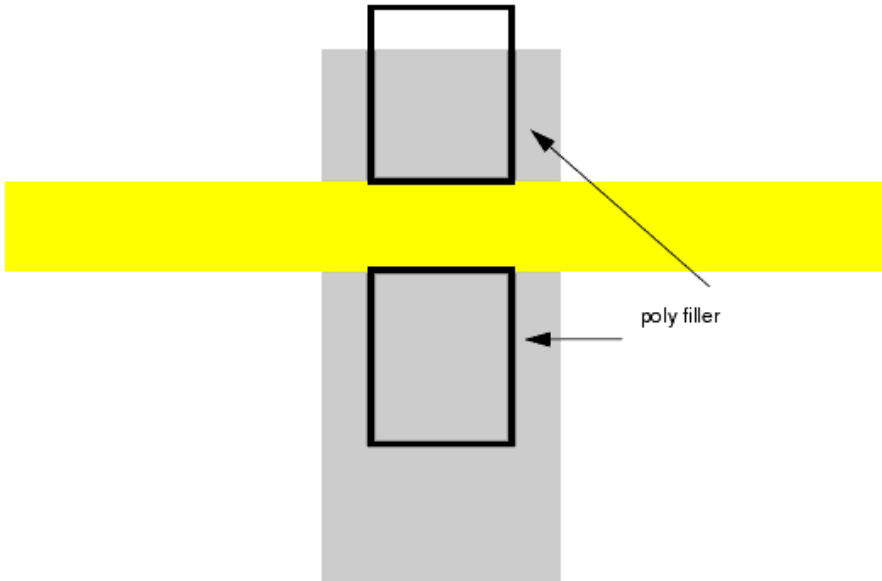


So far these rules have identified the new polysilicon for the channel and for the big extensions, and have discarded small extensions. But the new gate does not have the required extension. Re-build the 2 micron extension at the new channel length.

```
lyrOldGate = dve_bool_and(lyrPoly, lyrDiff);
lyrOversizedOldGate = dve_oversize(lyrOldGate, 2.0);
```

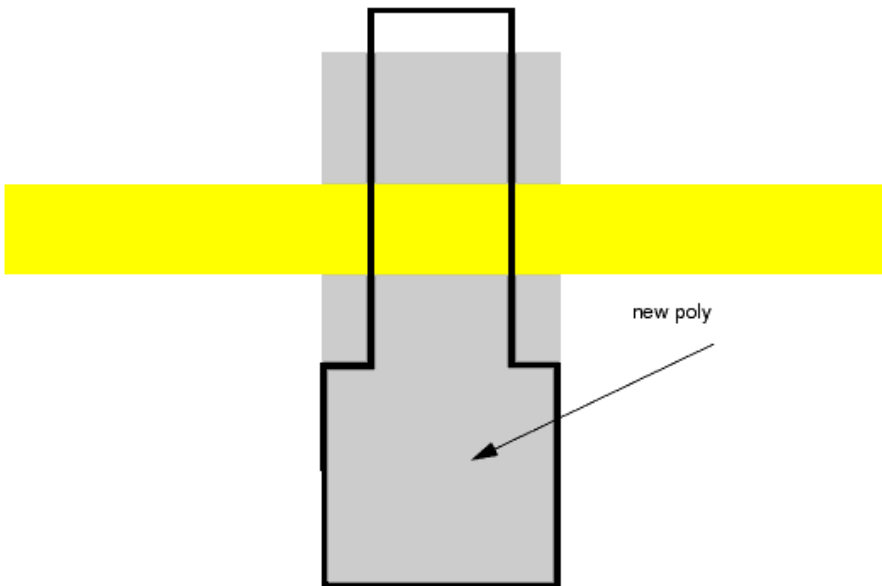
Then use a nest clearance rule to extract the result:

```
lyrNestEdges = dve_drc(nests(lyrNewGate, lyrOversizedOldGate) <= 2.0,
DVE_RN_TEMPLATE, DVE_RV_OPPOSITE);
lyrPolyFiller = dve_quadout(lyrNestEdges);
lyrError103 += dve_drc(all_edges(lyrPolyFiller), "lyrPolyFiller");
```

Finally, just assemble all the bits and pieces of compensated poly:

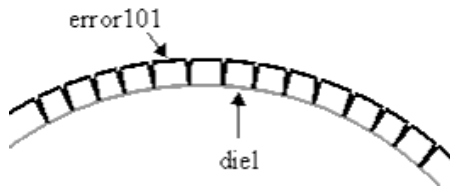
```
lyrNewPoly = dve_merge(lyrNewGate, lyrBigPolyExtension, lyrPolyFiller);
lyrError104 += dve_drc(all_edges(lyrNewPoly ), "lyrNewPoly ");
```



Example 3 - How to Avoid Spikes

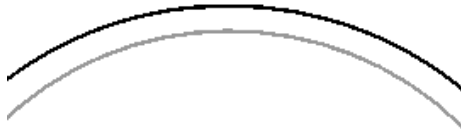
Consider the following use of the quadout operation on a circle:

```
decl lyrDiel = dve_import_layer("diel");
decl drcError = dve_export_layer("ads_drc_error");
decl lyrTmp1, lyrTmp2, lyrTmp3 = NULL;
lyrTmp1      = dve_drc (width (lyrDiel) < 200.0);
lyrTmp2      = dve_drc (compensate (lyrTmp1, 5.0));
lyrTmp3      = dve_quadout (lyrTmp2 );
drcError     = dve_bool_or (lyrDiel, lyrTmp3);
```



Spikes are created because the default template - OPPOSITE - is used. The solution is to select a different TEMPLATE: instead of the OPPOSITE template we will use an ARC template:

```
lyrTmp1 = dve_drc(width(lyrDie1) < 200.0, DVE_RN_TEMPLATE, DVE_RV_ARC);
```



Example 4 - Tips on Accelerating Quadout()

Instead of using the width command (with the problems of TEMPLATES), the command `all_edges` can be used:

```
lyrTmp1 = dve_drc(all_edges(diel));
lyrTmp2 = dve_drc(compensate (lyrTmp1, 1.0));
lyrTmp3 = dve_quadout (lyrTmp2);
drcError = dve_bool_or(diel, lyrTmp3);
```

This command is much faster.

Note

The option "Sort GEM layers" in the Verification tab of the Preferences dialog should not be checked when `all_edges` is used.

Polygon Extraction

This section describes the DRC command used for polyextraction from work layer to output DRC layer. These functions include:

- *dve_plg_extract()* (drc)

dve_plg_extract()

Extracts entire polygons from selected layer. Returns: A polygon layer.

Syntax

```
dve_plg_extract(workLayer);
```

where:

<i>workLayer</i>	A layer containing polygon data
------------------	---------------------------------

Example

```
decl lyrCond = dve_import_layer ("cond");  
decl lyrCond2 = dve_import_layer ("cond2");  
decl drcError = dve_export_layer ("ads_drc_error");  
decl lyrWork = NULL;  
lyrWork = dve_bool_and(lyrCond, lyrCond2);  
// Extract the polygon data from work layer to a DRC layer  
drcError += dve_plg_extract(lyrWork);
```

Polygon Selection

The section include topics related to:

- [Polygon Selection Based on Intrinsic Properties](#)
- *Polygon Selection Based on Merge Properties* (drc)
- *Polygon Selection Based on Edge Relationships* (drc)

Polygon Selection Based on Intrinsic Properties

Polygon Selection Based on Intrinsic Properties (output layer contains polygons) selection functions include:

- *poly_area()* (drc)
- *poly_hole_count()* (drc)
- *poly_line_length()* (drc)
- *poly_perimeter()* (drc)

poly_area()

Selects polygons based upon area. For polygons with holes, the area of the hole is subtracted. Returns: A polygon layer.

See also: *dve_drc()* (drc)

Syntax

`dve_drc (poly_area (inLayer) operator value);`

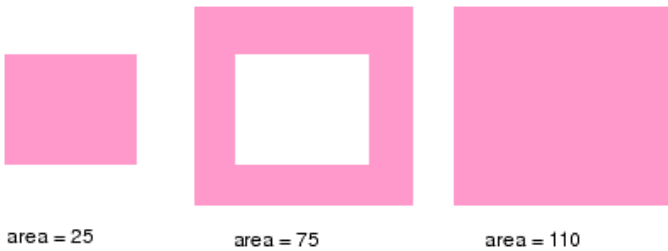
where:

<i>inLayer</i>	A polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>value</i>	A real value in layout units

Example

```
decl lyrCond = dve_import_layer("cond");
decl drcError = dve_export_layer("ads_drc_error");
decl lyrPoly=NULL;
lyrPoly = dve_drc(poly_area(lyrCond) < 100);
```

```
drcError += dve_drc(all_edges(lyrPoly), "Polygon area < 100");
```



poly_edge_code()

Select polygons based upon edge code information computed during a merge operation. Select only polygons with have all the given path types. Input layer must be the result of a merge command. Returns: A polygon layer.

See also: *dve_drc()* (drc)

Syntax

```
dve_drc (poly_edge_code (inLayer) [,qualifierName,qualifierValue]);
```

where:

<i>inLayer</i>	A polygon layer produced by a merge operation between two layers
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection

Selection Qualifier: DVE_RN_EDGE_SELECT

Qualifier Resource Value

<i>DVE_RV_ACCEPT_ANY</i>	Select polygon if any path codes are found
<i>DVE_RV_ACCEPT_ALL</i>	(default) Select polygon if all path codes are found
<i>DVE_RV_REJECT_ANY</i>	Reject polygon if any one of the path codes are found
<i>DVE_RV_REJECT_ALL</i>	Reject polygon if exactly all the path codes are found

If both accept and reject values are specified then a polygon passes the test only if it does have the edge codes specified in the accept command, and does not have the codes in the reject command.

Edge Code Qualifiers

DVE_RN_PATH_CODE (drc)

Example 1

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
```

```

decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPolyCombine = NULL;
decl lyrPolyOverlap = NULL;
decl lyrPolyMerge = NULL;
decl lyrPoly = NULL;
lyrPolyCombine = dve_bool_or (lyrCond, lyrCond2);
lyrPolyOverlap = dve_bool_and (lyrCond, lyrCond2);
lyrPolyMerge = dve_bool_and (lyrPolyCombine, lyrPolyOverlap);
lyrPoly = dve_drc (poly_edge_code (lyrPolyMerge),
    DVE_RN_EDGE_SELECT, DVE_RV_ACCEPT_ANY,
    DVE_RN_PATH_CODE, DVE_RV_INT);
drcError += dve_drc (all_edges (lyrPoly), "Conductive metal overlaps");

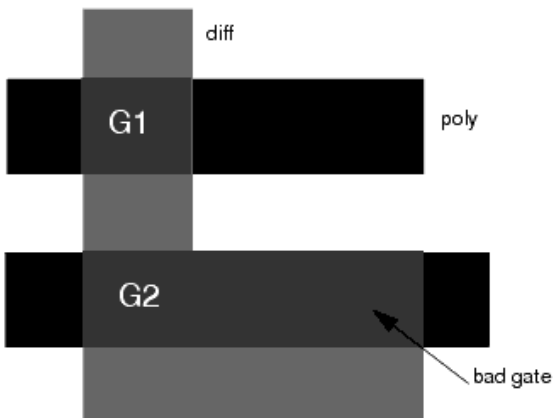
```

Example 2

Consider the recognition of a MOS gate:

```
decl lyrGate = dve_bool_and(lyrPoly, lyrDiff);
```

But suppose there is a section of the gate where a lyrPoly internally butts diffusion:



The problem is how to distinguish between gates G1 (legal) and G2 (illegal). The solution is to consider the edge codes that make up the polygon. Internally, a bit is set for each type of edge included in the polygon. So that

- Gate G1 has codes TIB and BIT
- Gate G2 has codes TIB, BIT, INT

The gates can be selected by accepting/rejecting polygons containing the edge code INT:

```

lyrGate = dve_bool_and(lyrPoly, lyrDiff);
lyrGoodGate = dve_drc(poly_edge_code(lyrGate), DVE_RN_EDGE_SELECT,
    DVE_RV_REJECT_ANY, DVE_RN_PATH_CODE, DVE_RV_INT);
lyrBadGate = dve_drc(poly_edge_code(lyrGate), DVE_RN_EDGE_SELECT,
    DVE_RV_ACCEPT_ANY, DVE_RN_PATH_CODE, DVE_RV_INT);
drcError += dve_drc(all_edges(lyrGoodGate ), "Good Gate");
lyrError102 += dve_drc(all_edges(lyrBadGate ), "Bad Gate");

```



poly_hole_count()

Selects polygons based upon the number of holes. Returns: A polygon layer.

See also: *dve_drc()* (*drc*)

Syntax

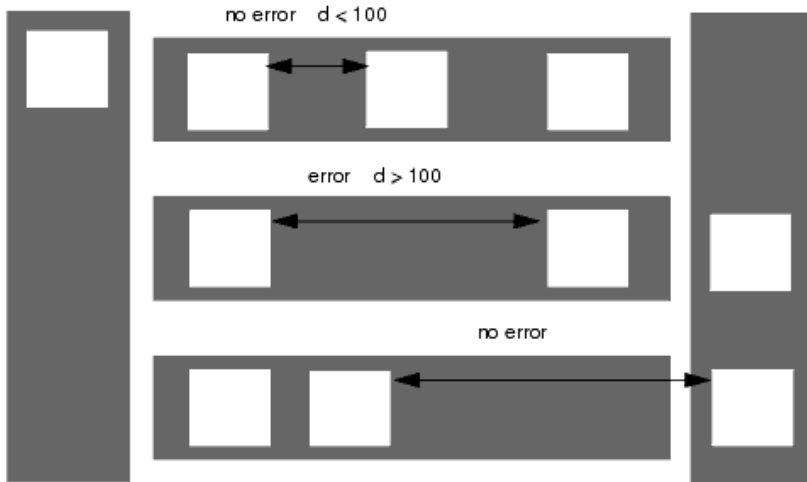
`dve_drc (poly_hole_count (inLayer) operator numHoles);`

where:

<i>inLayer</i>	A polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>numHoles</i>	An integer value representing the number of holes

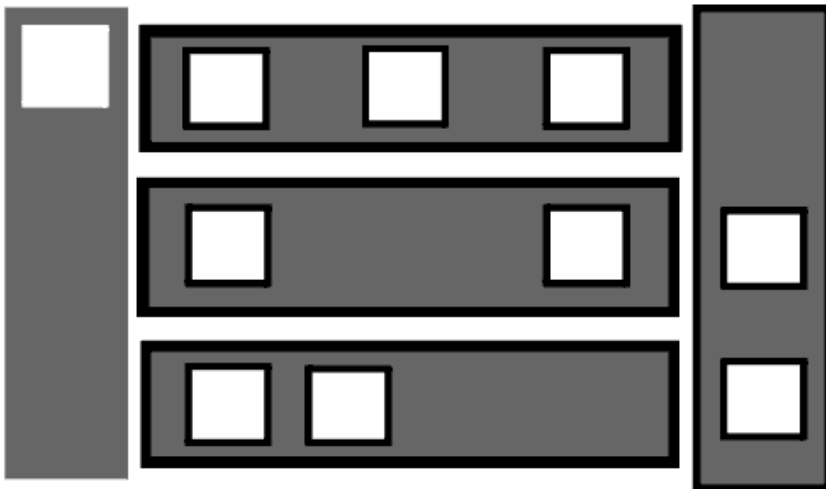
Example

Consider a clearance check which is applied only between polygons on layer cond2 that are contained in the same polygon on layer cond:



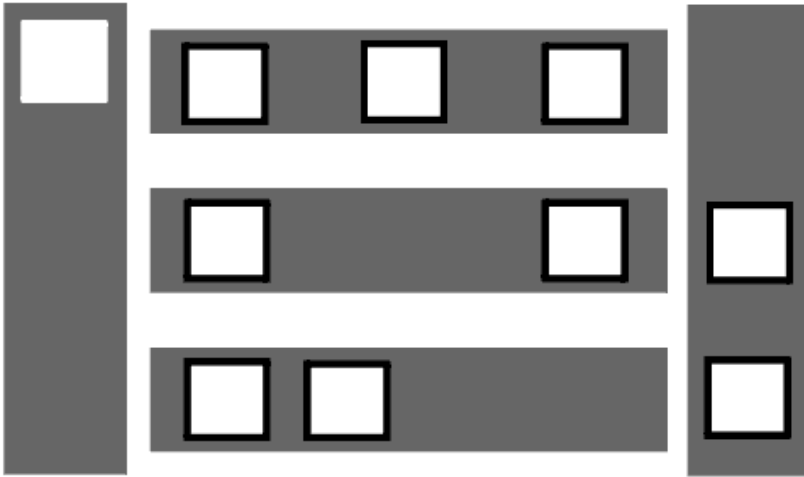
We create new polygons by subtracting layer cond2 from layer cond. Polygons with > 1 hole are selected using `poly_hole_count()`. Polygons with no or 1 hole are not selected because they contain no or 1 polygon on cond2.

```
lyrPoly = dve_bool_not(cond, cond2);
lyrPolyHole = dve_drc(poly_hole_count(lyrPoly) > 1);
```



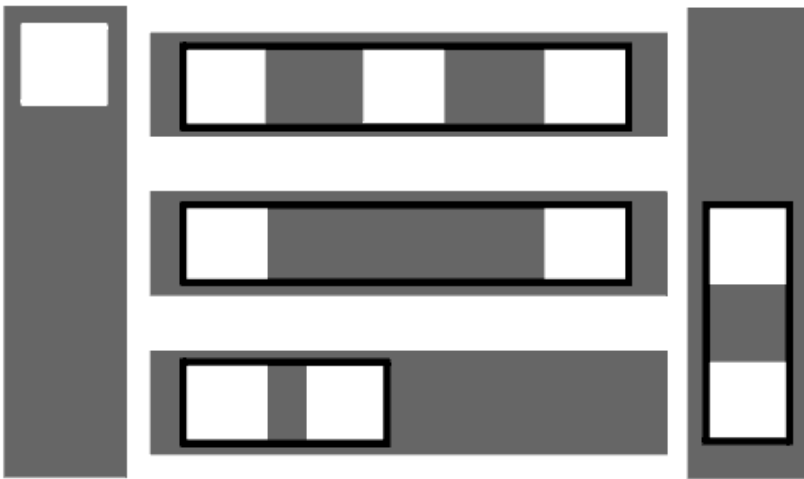
We use a clearance rule to select the inner edges. Qualifiers instruct the checker to direct the search between edges of same polygon and toward outside the polygon.

```
lyrEdges = dve_drc(single_clearance(lyrPolyHole) > 1, DVE_RN_EDGE_ANGLES,
DVE_RV_PARALLEL, DVE_RN_POLARITY, DVE_RV_OUTSIDE, DVE_RN_TEMPLATE,
DVE_RV_OPPOSITE, DVE_RN_STRUCTURE, DVE_RV_SAME_POLYGON);
```

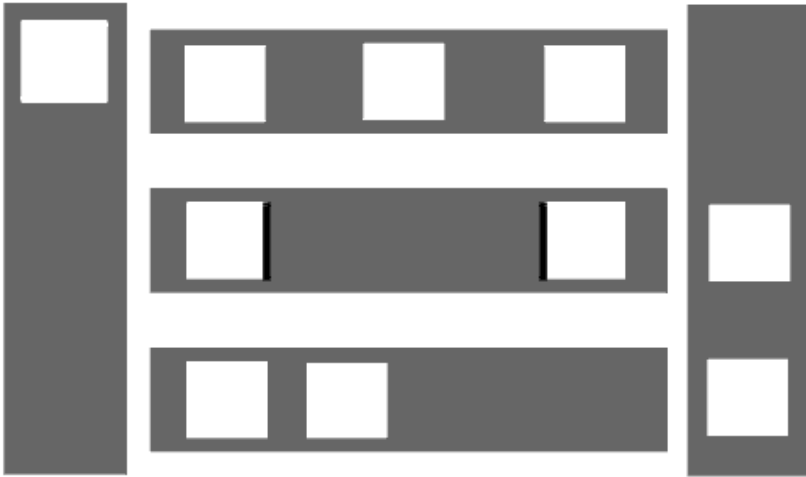
Polygons are extracted with the quadout command.

```
lyrPolyCmp = dve_quadout(lyrEdges);
```



Finally, cond2 is subtracted from the new polygon layer and a clearance rule is applied. This time, the search is directed toward inside the polygons.

```
lyrPoly = dve_bool_not(lyrPolyCmp, cond2);
error= dve_drc(single_clearance(lyrPoly )> 100, "spacing between cond2 and
cond2 inside cond is > 100.0um", DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL,
DVE_RN_POLARITY, DVE_RV_INSIDE, DVE_RN_TEMPLATE, DVE_RV_OPPOSITE,
DVE_RN_STRUCTURE, DVE_RV_SAME_POLYGON);
```



poly_inter_layer()

Select polygons on one layer (*inLayer1*) in relation to edges of polygons on another layer (*inLayer2*) if any of the given constrains are true. Returns a polygon layer.

See also: *dve_drc()* (*drc*)

Syntax

```
dve_drc (poly_inter_layer (inLayer1, inLayer2) [, qualifierName, qualifierValue]);
```

where:

<i>inLayer1, inLayer2</i>	A polygon layers
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the selection


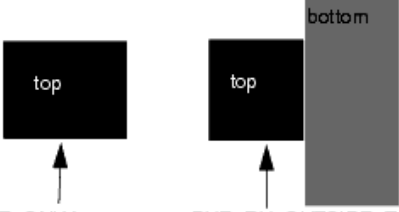
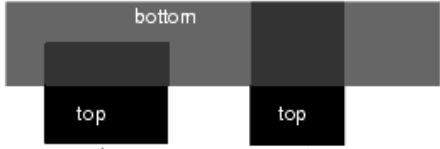

Selection Qualifier: DVE_RN_INTER_SELECT

Qualifier Resource Value

<i>DVE_RV_ACCEPT</i>	Select polygon if any path codes are found
<i>DVE_RV_REJECT</i>	Reject polygon if any one of the path codes are found

Poly Code Qualifiers: DVE_RN_INTER_CODE

Qualifier Resource Value

<i>DVE_RV_INSIDE_ONLY</i>	(default) The contained top is completely inside bottom and does not touch the inside of bottom.
<i>DVE_RV_INSIDE_TOUCH</i>	The contained top does touch the inside of bottom.
<i>DVE_RV_INSIDE</i>	DVE_RV_INSIDE_ONLY or DVE_RV_INSIDE_TOUCH  DVE_RV_INSIDE_ONLY DVE_RV_INSIDE_TOUCH
<i>DVE_RV_OUTSIDE_ONLY</i>	Top is completely outside bottom and does not touch the outside of bottom
<i>DVE_RV_OUTSIDE_TOUCH</i>	Top does touch the outside of bottom
<i>DVE_RV_OUTSIDE</i>	DVE_RV_OUTSIDE_ONLY or DVE_RV_OUTSIDE TOUCH  DVE_RV_OUTSIDE_ONLY DVE_RV_OUTSIDE_TOUCH
<i>DVE_RV_CUT_ONLY</i>	Top is partly inside bottom and partly outside bottom with no internal butt with bottom, that is, it does not touch the inside of bottom
<i>DVE_RV_CUT_TOUCH</i>	Top is partly inside bottom and partly outside bottom and does internal butt (touch) bottom
<i>DVE_RV_CUT_ANY</i>	DVE_RV_CUT_ONLY, DVE_RV_CUT_TOUCH  DVE_RV_CUT_ONLY DVE_RV_CUT_TOUCH
<i>DVE_RV_ENCLOSE_ONLY</i>	The contained bottom is completely inside top, and does not touch the inside of top
<i>DVE_RV_ENCLOSE_TOUCH</i>	The contained bottom does touch the inside of top
<i>DVE_RV_ENCLOSE</i>	DVE_RV_ENCLOSE_ONLY or DVE_RV_ENCLOSE_TOUCH
<i>DVE_RV_CUT</i>	DVE_RV_CUT_ANY, DVE_RV_ENCLOSE  DVE_RV_ENCLOSE_ONLY DVE_RV_ENCLOSE_TOUCH

Example

```

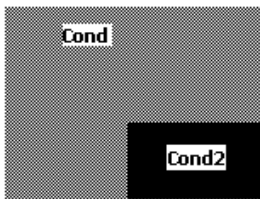
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = NULL;

```

```
lyrPoly = dve_drc (poly_inter_layer (lyrCond, lyrCond2),
    DVE_RN_INTER_CODE, DVE_RV_OUTSIDE);
drcError += dve_drc (all_edges (lyrPoly), "Conductive metal outside");
lyrPoly = dve_drc (poly_inter_layer (lyrCond, lyrCond2),
    DVE_RN_INTER_SELECT, DVE_RV_REJECT,
    DVE_RN_INTER_CODE, DVE_RV_OUTSIDE_TOUCH,
    DVE_RN_INTER_CODE, DVE_RV_INSIDE_TOUCH);
drcError += dve_drc (all_edges (lyrPoly),
    "Conductive metal outside and inside touch");
```

Example 2

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = NULL;
lyrPoly = dve_drc (poly_inter_layer (lyrCond, lyrCond2),
    DVE_RN_INTER_CODE, DVE_RV_ENCLOSE_TOUCH);
drcError += dve_drc (all_edges (lyrPoly), "Conductive metal enclose touch");
```



poly_line_length()

Selects polygons based upon the minimum line length. Returns: A polygon layer.

See also: *dve_drc()* (drc)

Syntax

```
dve_drc (poly_line_length (inLayer) operator distance [, qualifierName,
qualifierValue]);
```

where:

<i>inLayer</i>	A polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>value</i>	A real value in layout units
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the rule

Line Length Resource Qualifier: DVE_RN_LINE_LENGTH

Qualifier Resource Value

DVE_RV_MIN_LINE	(default) Select based upon minimum line length of polygon
DVE_RV_MAX_LINE	Select based upon maximum line length of polygon

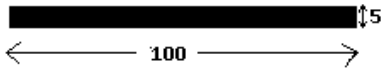
Examples

1. Check based on the maximum line length.

```
decl lyrCond = dve_import_layer ("cond");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = NULL;
lyrPoly = dve_drc (poly_line_length (lyrCond) <= 10.0,
    DVE_RN_LINE_LENGTH, DVE_RV_MAX_LINE);
drcError += dve_drc (all_edges (lyrPoly), "Polygon length < 10.0");
```

Check based on the minimum line length.

```
decl lyrCond = dve_import_layer ("cond");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = NULL;
lyrPoly = dve_drc (poly_line_length (lyrCond) <= 20.0,
    DVE_RN_LINE_LENGTH, DVE_RV_MIN_LINE);
drcError += dve_drc (all_edges (lyrPoly), "Polygon length < 20.0");
```



For example 1, None of the above polygon gets selected. As the check is applicable for maximum line length, so both 100 and 50 is greater 10.

For example 2, Both of them gets selected. As check also takes into consideration the minimum line length that is 5.

poly_path_count()

Select polygons based upon path count information computed during a merge operation. Input layer must be the result of a merge command. Returns: A polygon layer.

See also: *dve_drc()* (drc)

Syntax

```
dve_drc (poly_path_count (inLayer) operator distance [, qualifierName, qualifierValue]);
```

where:

<i>inLayer</i>	A polygon layer produced by a merge operation between two layers
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>value</i>	A real value in layout units
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the rule

Path Count Qualifier: DVE_RN_PATH_COUNT

Qualifier Resource Value:

<i>DVE_RV_PATH_COUNT</i>	(default) Select based upon path count of top polygon
<i>DVE_RV_ANTI_PATH_COUNT</i>	Select based upon path count of bottom polygon

Path Code Qualifiers

DVE_RN_PATH_CODE (drc)

Example 1

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPolyMerge = NULL;
decl lyrPoly = NULL;
lyrPolyMerge = dve_bool_not (lyrCond, lyrCond2);
lyrPoly = dve_drc (poly_path_count (lyrPolyMerge) >= 1,
    DVE_RN_PATH_CODE, DVE_RV_TOP,
    DVE_RN_PATH_CODE, DVE_RV_INT);
drcError += dve_drc (all_edges (lyrPoly),
    "Metal layer outside and butting internally");
```

Example 2

See `poly_path_length()`, *Example 2* (drc)

poly_path_length()

Select polygons based upon path length properties computed during a merge operation. Input layer must be the result of a merge command. Returns: A polygon layer.

See also: `dve_drc()` (drc)

Syntax

```
dve_drc (poly_path_length (inLayer) operator distance [qualifierName, qualiferValue]);
```

where:

<i>inLayer</i>	A polygon layer produced by a merge operation between two layers
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>value</i>	A real value in layout units
<i>qualifierName, qualifierValue</i>	A name, value pair that qualifies the rule

Path Type Qualifier: DVE_RN_PATH_LENGTH

Qualifier Resource Value

<i>DVE_RV_MIN_PATH</i>	Select based upon minimum path length of top polygon
<i>DVE_RV_MAX_PATH</i>	Select based upon maximum path length of top polygon
<i>DVE_RV_TOTAL_PATH</i>	Select based upon total path length of top polygon
<i>VE_RV_MIN_ANTI_PATH</i>	Select based upon minimum path length of bottom polygon
<i>DVE_RV_MAX_ANTI_PATH</i>	Select based upon maximum path length of bottom polygon
<i>DVE_RV_TOTAL_ANTI_PATH</i>	Select based upon total path length of bottom polygon

Path Code Qualifiers

DVE_RN_PATH_CODE (drc)

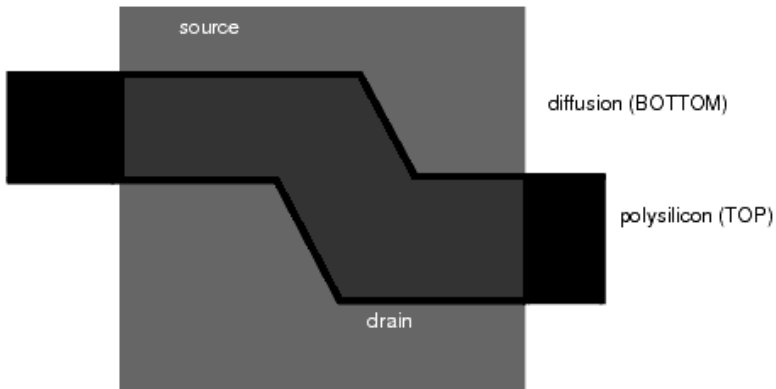
Example 1

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPolyMerge = NULL;
decl lyrPoly = NULL;
lyrPolyMerge = dve_bool_not (lyrCond, lyrCond2);
lyrPoly = dve_drc (poly_path_length (lyrPolyMerge) < 20.0,
  DVE_RN_PATH_CODE, DVE_RV_TOP,
  DVE_RN_PATH_LENGTH, DVE_RV_MIN_PATH);
drcError += dve_drc (all_edges (lyrPoly),
  "Polygon path length < 20.0");
```

Example 2

Consider the rule to extract a transistor channel formed by the overlap of polysilicon and diffusion:

```
lyrChannel = dve_bool_and(lyrPoly, lyrDiff);
```



The edge codes can be used to determine the width and the length of the transistor channel. The ends (source and drain) of the channel are formed by TIB edges. The sides of the channels are formed by BIT edges. In order to measure the width of the channel, define which edges of the polygon constitute the path of interest. In this case, specify that the path is made up of TIB edges:

```
dve_drc(...DVE_RN_PATH_CODE, DVE_RV_TIB,...);
```

To check the channel width as path length grater than 10:

```
lyrEnds = dve_drc(poly_path_length(lyrChannel) > 10, DVE_RN_PATH_CODE,
DVE_RV_TIB, DVE_RN_PATH_LENGTH, DVE_RV_MAX_PATH);
drcError += dve_drc(all_edges(lyrSides ), "long channel");
```

To select 'thin' channels :

```
lyrSides = dve_drc(poly_path_length(lyrChannel) < 2, DVE_RN_PATH_CODE,
DVE_RV_TIB, DVE_RN_PATH_LENGTH, DVE_RV_MIN_ANTI_PATH);
drcError += dve_drc(all_edges(lyrSides), "thin channel");
```

Badly formed channels can be rejected using poly_path_count():



```
lyrGoodChannel = dve_drc(poly_path_count(lyrChannel) == 2, DVE_RN_PATH_CODE,
DVE_RV_TIB, DVE_RN_PATH_COUNT, DVE_RV_PATH_COUNT);
```

This rule means that you can accept only good channels when the number of TIB paths equal 2.

Polygon Selection Based on Edge Relationships

Polygon Selection Based on Edge Relationships (output layer contains polygons) selection

function includes: *poly_inter_layer()* (drc).

poly_perimeter()

Selects polygons based upon the total length of the outside edges. Returns: A polygon layer.

See also: *dve_drc()* (drc)

Syntax

dve_drc (*poly_perimeter* (*inLayer*) *operator* *distance*);

where:

<i>inLayer</i>	A polygon layer
<i>operator</i>	< Less than <= Less than or equal to == Equal to > Greater than >= Greater than or equal to
<i>distance</i>	A real value in layout units

Example

```
decl lyrCond = dve_import_layer ("cond");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = NULL;
lyrPoly = dve_drc (poly_perimeter (lyrCond) < 100.0);
drcError += dve_drc (all_edges (lyrPoly), "Polygon perimeter < 100.0");
```

For the above polygons,

In the first case, the perimeter of the polygon is 250 which is greater than 100. Hence the polygon doesn't get selected.

In the second case, the perimeter of the polygon is 75 which is less than 100. Hence the polygon gets selected.

Polygon Selection Based on Merge Properties

Polygon Selection Based on Merge Properties selection functions include:

poly_edge_code() (drc), *poly_path_count()* (drc), and *poly_path_length()* (drc).

Polygon merge qualifier commands constrain the selection of edges based upon a specified edge code. All of the commands in this section are based upon edge information computed during a merge operation.

Convention for TOP and BOTTOM layers

Some of the notation in this section depends on a naming convention. The first mentioned layer is called the TOP layer and the second mentioned layer is called the BOTTOM layer. For example,

```
lyrPoly = dve_bool_and(cond, cond2);
```

Layer cond is TOP, layer cond2 is BOTTOM.

This is an arbitrary, rather than a descriptive designation. For example the TOP layer might be "metal" and the BOTTOM layer might be "contact".

Using edge codes

When polygon TOP and polygon BOTTOM merge, a set of vertices consisting of the intersection points is derived. Each resultant edge between pairs of these vertices has a unique 'edge_code' that describe its relationship to other edges.

Consider two polygons on the TOP and BOTTOM levels (vertices at the intersection are shown as asterisks '*'):

```
TTTTTTTTTTTTTTTT polygon TOP
T                T
*bbbbbbbbbbbbbb*BBBBBBBBBBBB polygon BOTTOM
I                t                B
I                t                B
*bbbbbbbbbbbbbb*BBBBBB        B
T                T                B    B
T                T                B    B
T                *BBBBBB        B
T                E                B
T                E                B
T                *BBBBBBBBBBBB
T                T
TTTTTTTTTTTTTTTT
```

Merging the two polygons produces a merged database containing six types of edges. Each type is shown with a different character:

TOP_OUTSIDE_BOTTOM (T)	polygon TOP outside polygon BOTTOM
BOTTOM_OUTSIDE_TOP (B)	polygon BOT outside polygon TOP
TOP_INSIDE_BOTTOM (t)	polygon TOP inside polygon BOTTOM
BOTTOM_INSIDE_TOP (b)	polygon BOT inside polygon BOTTOM
INTERNAL (I)	edges of TOP and BOTTOM butting internally
EXTERNAL (E)	edges of TOP and BOTTOM butting externally

A side effect of the algorithm is that corner-to-edge and corner-to-corner interactions are not evaluated. For instance, when checking to see if two polygons touch, a contact between a corner and an edge, or between two corners is not counted.

For example, these polygons are not classified as touching:



Edge Code Qualifier

The following qualifier is used extensively in polygon selection commands:

DVE_RN_PATH_CODE

Qualifier Resource Value:

<i>DVE_RV_TOP</i>	(default) Select edges on top that are outside bottom
<i>DVE_RV_BOT</i>	Select edges on bottom that are outside top
<i>DVE_RV_TIB</i>	Select edges on top that are inside bottom
<i>DVE_RV_BIT</i>	Select edges on bottom that are inside top
<i>DVE_RV_INT</i>	Select edges on top and bottom that are butting internally
<i>DVE_RV_EXT</i>	Select edges on top and bottom that are butting externally

Definition of paths and anti-paths

A "path" is a set of coincident edges of a polygon, all of the same type. Other paths of the polygon which do not satisfy the requested path type are called the "anti-paths" of the polygon.

Sizing Operations on Polygons

This section describes the DRC commands used for sizing operations on polygons. These commands include:

- *dve_oversize()* (drc)
- *dve_undersize ()* (drc)

dve_oversize()

Moves edges of polygons by the given sizing distance. All edges are moved in parallel toward outside of polygons.

Syntax

```
dve_oversize(inLayer, size);
```

where:

<i>inLayer</i>	A polygon layer
<i>size</i>	A real value in Layout units of the amount by which the size is to be increased

Example

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = NULL;
decl lyrOversize = NULL;
lyrPoly = dve_drc (poly_inter_layer (lyrCond, lyrCond2),
  DVE_RN_INTER_CODE, DVE_RV_OUTSIDE);
drcError += dve_drc (all_edges (lyrPoly), "Conductive metal outside");
lyrOversize = dve_oversize(lyrPoly, 5);
drcError += dve_drc (all_edges (lyrOversize), "Oversize Conductive metal
outside");
```



dve_undersize ()

Moves edges of polygons by the given sizing distance. All edges are moved in parallel toward inside of polygons.

Syntax

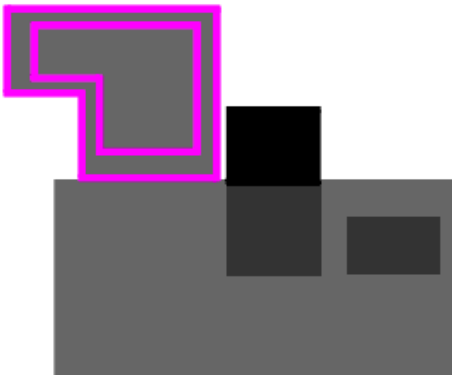
```
dve_undersize(inLayer, size);
```

where:

<i>inLayer</i>	A polygon layer
<i>size</i>	A real value in Layout units of the amount by which the size is to be decreased

Example

```
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl drcError = dve_export_layer ("ads_drc_error");
decl lyrPoly = NULL;
decl lyrUndersize = NULL;
lyrPoly = dve_drc (poly_inter_layer (lyrCond, lyrCond2),
    DVE_RN_INTER_CODE, DVE_RV_OUTSIDE);
drcError += dve_drc (all_edges (lyrPoly), "Conductive metal outside");
lyrUndersize = dve_undersize(lyrPoly, 5);
drcError += dve_drc (all_edges (lyrUndersize), "Undersize Conductive metal
outside");
```



Troubleshooting Design Rule Checker

If a `dve_drc` command is not producing the expected output, try the following debugging techniques:

- Resolve any compile errors or warnings.
- Check to make sure the `dve_drc` command has an error message.
- If possible, always use `<` for clearance rules to ensure a bounded check.
- Inspect the input layers using the layer editor. Send the data to an export layer (be sure to include an error message), and view the data using *Load Results*.

Layer Management Errors (101-199)

101 Import layer must be a design layer

Import and export layers must be defined as physical design layers

102 Export layer must be a design layer

Import and export layers must be defined as physical design layers

103 No output layer

An output layer is required on the left-hand side of the equal sign (=).

104 Layer parameter is uninitialized

Input layers must have previously appeared on the left-hand side of an equal sign (=).

105 Layer parameter is an export layer

An input layer that has been declared as an export layer cannot be used as an input layer.

106 No import layers defined

At least one input layer must be defined.

107 No export layers defined

At least one export layer must be defined.

108 Rules do not generate output

At least one rule must assign data to an export layer.

Layer Management Warnings (201-299)

201 Redefining an import layer

A layer that has been declared as an import layer is being redefined.

202 Redefining an export layer

A layer that has been declared as an export layer is being redefined.

Command Usage Errors (301-399)

301 Expecting layer parameter

Parameter is uninitialized or is not a layer. Please see documented command syntax.

302 Expecting a string parameter

Parameter is uninitialized or is not a string. Please see documented command syntax.

303 Expecting an integer parameter

Parameter is uninitialized or is not an integer number. Please see documented command syntax.

304 Expecting a real parameter

Parameter is uninitialized or is not a real number. Please see documented command syntax.

305 Invalid angle parameter

Expecting a real number greater than 0 and less than 360 with only one decimal point of precision.

306 Command is a dve_drc subfunction

Command must appear as the first parameter to a dve_drc subfunction.
Command is not valid outside the context of a dve_drc command.

307 Unsupported operator

The dve_drc expression contains an unrecognized operator. Valid operators are

<	Less than
<=	Less than or equal to
==	Equal to
>	Greater than
>=	Greater than or equal to

308 Unsupported set operator

The command is missing the left-hand equal sign for assignment to an output layer.

309 Missing elements of expression

The command requires an expression. Please see the documented command syntax.

310 Expecting polygon layer

Polygon layers are produced as the result of polygon selection or boolean commands. Edge operation commands perform segment merging, sizing and polygon extraction on selected edges.

311 Expecting edge layer

Edge layers are produced as the result of an edge selection, edge compensation, or edge operation command. Polygon commands perform polygon selection and boolean operations on polygons.

312 Expecting boolean merge layer

Polygon selection commands based on merge properties only accept input layers that are the direct result of a boolean polygon merge operation such as `dve_bool_and`.

313 Expecting dve_drc subfunction

The `dve_drc` command must always appear with a `dve_drc` subfunction as the first parameter.

314 Nested merge not allowed

The result of a `dve_merge` command cannot be used as the input to another `dve_merge` command.

315 Invalid use of compensate layer

Output layer of `compensate` can only be used as input to `dve_plgout` and `dve_quadout` commands.

Command Usage Warnings (401-499)

401 Expression ignored

Command does not require an expression

402 Qualifier ignored

Resource qualifiers that do not apply are ignored. Please see documented command syntax.

403 Using default polarity

A polarity specification is required for commands `double_clearance` and `single_clearance`. If no polarity is specified, a polarity `DVE_RV_OUTSIDE` is used.

404 Using default template

A template specification is required for commands `double_clearance` and `single_clearance`. If no template is specified, a template `DVE_RV_OPPOSITE` is

used.

405 Clearance qualifiers ignored

Clearance qualifiers require an upper bound and are currently not supported for unbounded greater-than (>) or greater-than-or-equal (>=). This can be corrected by using range comparisons.

406 Using layer name as message string

No message output is specified in the command. A default message is generated based on the output layer name.

407 There are rules larger than the circuit size

Rules much larger than the circuit may cause the DRC engine to enter into a very long loop to check the circuit and clean up the temporary files.

Using Calibre DRC Link

You can use Calibre DRC to check your layout against design rules by choosing the Calibre DRC engine (a Calibre rule file is required). When you choose the Calibre DRC engine, ADS exports your layout design to GDSII, writes a Calibre DRC control file, executes a design rule check in Calibre, and displays the ASCII-formatted Calibre DRC results in the ADS DRC results viewer.

ADS-Calibre DRC Link runs in two modes: *Local* and *Remote*. Use *Local* if ADS and Calibre are available on the same Linux or Solaris machine. Use *Remote* if you are running ADS on a Windows, Linux, or Solaris machine and have network access to Calibre installed on a different Linux or Solaris machine.

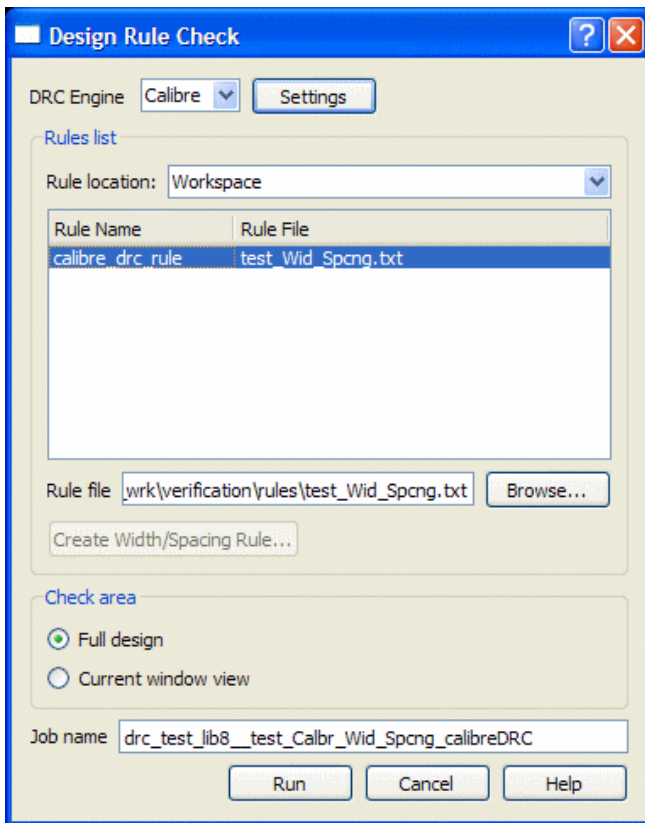
Running Calibre DRC in Local Mode

To prepare your Linux or Solaris machine:

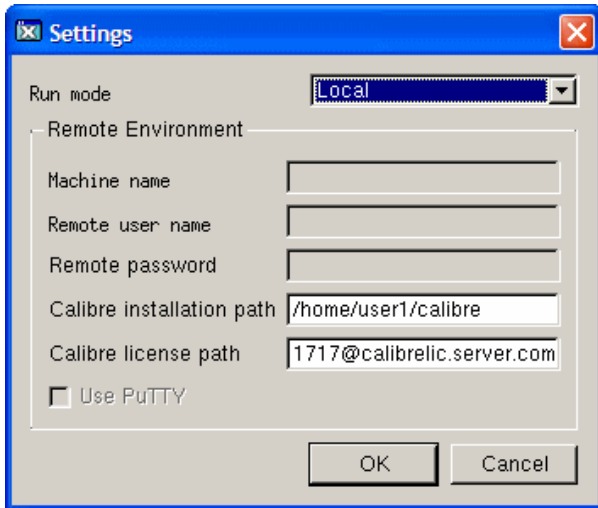
1. Declare the `MGLS_LICENSE_FILE` environment variable.
2. Declare the `MGC_HOME` environment variable.

To run Calibre DRC in Local Mode:

1. From your Layout window, click **Tools > DRC**.
2. Choose **Calibre** as the *DRC Engine* and click **Settings** in the Design Rule Check dialog box.



3. Choose **Local** for the *Run mode* in the Calibre DRC Link dialog box.



- 3.
4. You may enter the Calibre installation and license path values. In such cases the values entered in the dialog would take precedence over the corresponding values set in the `MGLS_LICENSE_FILE` and `MGC_HOME` variable.
5. Click **OK** in the Calibre DRC Link dialog box to return to the Design Rule Check dialog box.
6. Click **Browse** to choose the **Rule file** and click **OK**.
7. Click **Run** to execute the Calibre DRC.

Running Calibre DRC in Remote Mode

In remote mode, Calibre DRC is run on a remote machine using a secure shell.

To prepare your remote machine

1. Declare the `MGLS_LICENSE_FILE` environment variable.
2. Declare the `MGC_HOME` environment variable. Alternatively, `MGC_HOME` can be set by specifying Calibre installation path in the settings dialog. If `MGC_HOME` is specified in the settings dialog then it is not read from the system environment.
3. Declare the `DRC_MAPPED_DRIVE` environment variable. Mount the home directory of the remote UNIX machine onto your local machine and set the value of `DRC_MAPPED_DRIVE` to the mount path. For example if you have mounted your remote **/home/user1** onto the Z: drive of your Windows machine, then set `DRC_MAPPED_DRIVE` to **Z:**. Setting the variable `DRC_MAPPED_DRIVE` is optional. If it is not set, FTP would be used to transfer files between the local and remote machine. Therefore, the remote FTP port should not be blocked by the administrator.
4. Ensure that secure shell, ssh is available and configured in the local and remote machine. For any user, it requires one time setup to configure ssh. ssh configuration requires generation of a private-public key pair with the below steps.

Note

In the settings dialog, the field for which user input is mandatory has bold font.

Generating a Private-Public Key Pair in Windows

1. Click **Start > Run** and enter `cmd` to open a DOS prompt dialog box.
2. Enter `echo %HOME%` to check your home directory.
Ensure that the home directory is the same as the home directory set in ADS. ADS home directory can be confirmed by running the command `de_info(getsystem("HOME"))`; in ADS command line tool (**Tools > Command Line**).
3. Enter `cd %HOME%` to go to home directory.
4. Create the directory `.ssh` in the home directory by entering

```
mkdir .ssh
```

5. Generate a private-public key pair by entering


```
ssh-keygen -t dsa -f $HOME/.ssh/id_dsa -P ''
```

If you get the message for `ssh-keygen` not found then include `%HPEESOF_DIR/tools/bin` in your `PATH`.

This should result in two files, `id_dsa` (private key) and `id_dsa.pub` (public key).

Generating a Private-Public Key Pair in Solaris/Linux

1. Enter `echo $HOME` to check your home directory.
Ensure that the home directory is the same as the home directory set in ADS. ADS home directory can be confirmed by running the command `de_info(getsysenv("HOME"))`; in ADS command line tool (**Tools > Command Line**).
2. Enter `cd $HOME` to go to home directory.
3. Create the directory `.ssh` in the home directory by entering


```
mkdir .ssh
```
4. Generate a private-public key pair by entering


```
ssh-keygen -t dsa -f $HOME/.ssh/id_dsa -P ''
```

This should result in two files, `id_dsa` (private key) and `id_dsa.pub` (public key).

Copy the public key to the remote machine:

1. Copy file `id_dsa.pub` from local machine to the remote machine.
2. On the remote machine run the following commands:

```
$ "cat id_dsa.pub >> $HOME/.ssh/authorized_keys"
$ "chmod 0600 $HOME/.ssh/authorized_keys"
```

Depending on the version of OpenSSH, the file `authorized_keys2` may also be required:

```
$ "cat id_dsa.pub >> $HOME/.ssh/authorized_keys2"
$ "chmod 0600 $HOME/.ssh/authorized_keys2"
```

An alternative is to create a link from `authorized_keys` to `authorized_keys2`:

```
$ "cd $HOME/.ssh "
$ "ln -s authorized_keys authorized_keys2"
```

On the local (client) machine, test the results by connecting to the remote (server) machine:

```
$ "ssh -i $HOME/.ssh/id_dsa <remote_machine_name>"
```

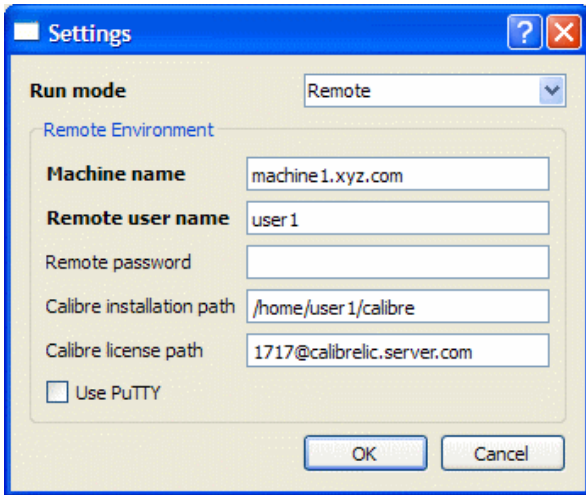
When you login the first time, you may be asked for confirmation yes/no. Enter `yes` to proceed. If everything is correct, you can login without any password to the remote machine.

If there is an issue, you may need to consult your system administrator.

To run Calibre DRC in Remote Mode

1. From your Layout window, click **Tools > DRC**.

2. Choose **Calibre** as the *DRC Engine* and click **Settings** in the Design Rule Check dialog box.
3. Choose **Remote** for the **Run mode** in the Calibre DRC Link dialog box.



4. Enter the **Machine name** and the **Remote user name**.
5. You may enter the Calibre installation and license path values. In such cases the values entered in the dialog would take precedence over the corresponding values set in the `MGLS_LICENSE_FILE` and `MGC_HOME` variable.
6. Click **OK** in the Calibre DRC Link dialog box to return to the Design Rule Check dialog box.
7. Click **Browse** to choose the *Rule file* and click **OK**.
8. Click **Run** to execute the Calibre DRC.

Note

1. If you do not wish to generate the SSH Private/Public Keys, then you can click the "Use PuTTY" option for ADS running on Windows machine. In such cases, you will be prompted to enter the password every time you run a LVS job. The PuTTY installation path **MUST** be specified in your Windows `PATH` environment variable.
 1. If you would like certain environment variables (for example, variables that are specific to your DRC Rule deck) to be made available to Calibre, you may do so by defining them in a file `calibre_config.sh`. This file **MUST** be kept inside your ADS workspace folder. If ADS finds this file while running Calibre DRC, it would first source this file on the remote UNIX machine before actually invoking Calibre. This provides you the flexibility to do any customization (for example: defining Rule Deck specific environment variables, copying other rules deck files etc) in the file `calibre_config.sh`

Viewing Calibre DRC Link Results

After the Calibre DRC results are available, ADS translates the Calibre results file to ADS readable format, displays Calibre DRC results in the **ADS DRC Results Viewer** and highlights the errors in ADS layout design.

The screenshot displays the ADS Desktop Design Rule Checker interface. The main window shows a layout titled "drc_test_lib8:test_Calbr_Wid_Spcng:layout * (Layout):1". The interface includes a menu bar (File, Edit, Select, View, Insert, Options, Tools, Schematic, EM, Window, DesignGuide, Help), a toolbar with various design tools, and a palette on the left containing numerous components like Maclin, MSABND, MBstub, Mcln, Mcorn, Mrozo, Mcurve, MGap, MICAP1-4, Mlang, and MLEF. The layout area shows a black background with a cyan rectangle on the left and a red rectangle on the right. A "DRC Results Viewer" window is open in the foreground, displaying the following information:

Design: drc_test_lib8:test_Calbr_Wid_Spcng:layout
 Job name: drc_test_lib8_test_Calbr_Wid_Spcng_calibreDRC

Current Fixed

Error	X	Y
[-] Minimum Gate width is 4.0		
1	-240.903	235.768
[+] Minimum Via2 width is 3.0		
[+] Minimum metal0 width is 4.5		
[+] Minimum spacing b/w Gate and Via2 is 2		
[+] Minimum spacing b/w Via2 and metal0 is 1		

Auto zoom Auto select

Design rule:
 Number of errors: 5

At the bottom of the interface, the status bar shows "Select: Enter the starting point", "0 items", "ads_annotate:ads_drawing1 -185.000, 215.000", and "um".

Using Assura DRC Link

You can use Assura to check your layout against design rules by choosing the Assura DRC engine (an Assura rule file is required). When you choose the Assura DRC engine, ADS uses the command line to export your layout design to GDSII, write an RSF input file, execute a design rule check in Assura, and display the ASCII-formatted Assura DRC results in the ADS DRC results viewer.

ADS-Assura DRC Link runs in two modes: *Local* and *Remote*. Use *Local* if ADS and Assura are available on the same Linux or Solaris machine. Use *Remote* if you are running ADS on a Windows, Linux, or Solaris machine and have network access to Assura installed on a different Linux or Solaris machine.

Note
ADS-Assura DRC link is supported for Assura version 3.2 onwards.

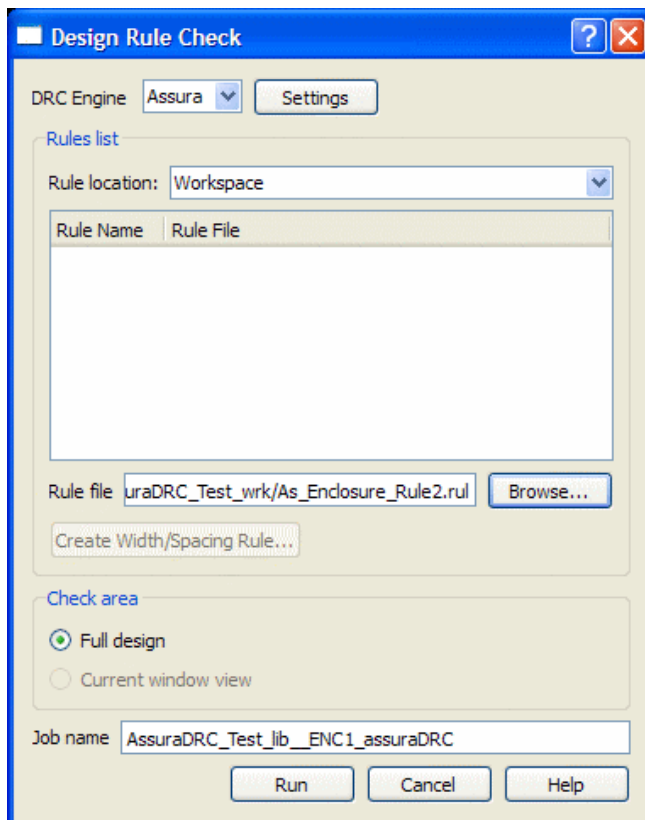
Running Assura DRC in Local Mode

To prepare your Linux or Solaris machine:

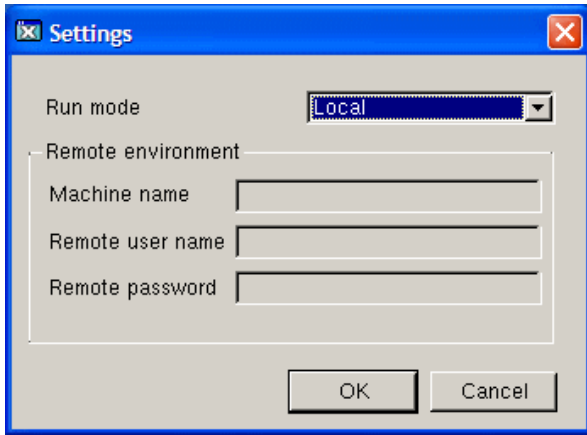
1. Declare the `ASSURA_LICENSE_FILE` environment variable.
2. Declare the `ASSURA_INSTALLATION_DIR` environment variable.

To run Assura DRC in Local Mode"

1. From your Layout window, click **Tools > DRC**.
2. Choose **Assura** as the *DRC Engine* and click **Settings** in the Design Rule Check dialog box.



3. Choose **Local** for the *Run mode* in the Assura DRC Link dialog box.



- 3.
4. Click **OK** in the Assura DRC Link dialog box to return to the Design Rule Check dialog box.
5. Click **Browse** to choose the *Rule file* and click **OK**.
6. Click **Run** to execute the Assura DRC.

Running Assura DRC in Remote Mode

In remote mode, Assura DRC is run on a remote machine using a secure shell.

To prepare your remote machine:

1. Declare the `ASSURA_LICENSE_FILE` environment variable.
2. Declare the `ASSURA_INSTALLATION_DIR` environment variable.
3. Declare the `DRC_MAPPED_DRIVE` environment variable. Mount the home directory of the remote UNIX machine onto your local machine and set the value of `DRC_MAPPED_DRIVE` to the mount path. For example if you have mounted your remote **/home/user1** onto the Z: drive of your Windows machine, then set `DRC_MAPPED_DRIVE` to **Z:**. Setting the variable `DRC_MAPPED_DRIVE` is optional. If it is not set, FTP would be used to transfer files between the local and remote machine. Therefore, the remote FTP port should not be blocked by the administrator.
4. Ensure that secure shell is available and configured in the remote machine. For any user, it requires one time setup to configure ssh. ssh configuration requires generation of a private-public key pair with the below steps.

Generating a Private-Public Key Pair in Windows

1. Click **Start > Run** and enter `cmd` to open a DOC prompt dialog box.
2. Enter `echo %HOME%` to check your home directory.
Ensure that the home directory is the same as the home directory set in ADS. ADS home directory can be confirmed by running the command `de_info(getsysenv("HOME"))`; in ADS command line tool (**Tools > Command Line**).
3. Enter `cd %HOME%` to go to home directory.
4. Create the directory `.ssh` in the home directory by entering
`mkdir .ssh`
5. Generate a private-public key pair by entering
`ssh-keygen -t dsa -f id_dsa -P ''`

If you get the message for `ssh-keygen not found` then include `%HPEESOF_DIR/tools/bin` in your PATH.

This should result in two files, `id_dsa` (private key) and `id_dsa.pub` (public key).

Generating a Private-Public Key Pair in Solaris/Linux

1. Enter `echo $HOME` to check your home directory.
Ensure that the home directory is the same as the home directory set in ADS. ADS home directory can be confirmed by running the command `de_info(getsysenv("HOME"))`; in ADS command line tool (**Tools > Command Line**).
2. Enter `cd $HOME` to go to home directory.
3. Create the directory `.ssh` in the home directory by entering
`mkdir .ssh`
4. Generate a private-public key pair by entering
`ssh-keygen -t dsa -f $HOME/.ssh/id_dsa -P ''`

This should result in two files, `id_dsa` (private key) and `id_dsa.pub` (public key).

Copy the public key to the remote machine:

1. Copy file `id_dsa.pub` from local machine to the remote machine.
2. On the remote machine run the following commands:

```
$ "cat id_dsa.pub >> $HOME/.ssh/authorized_keys"
$ "chmod 0600 $HOME/.ssh/authorized_keys"
```

Depending on the version of OpenSSH, the file `authorized_keys2` may also be required:

```
$ "cat id_dsa.pub >> $HOME/.ssh/authorized_keys2"
$ "chmod 0600 $HOME/.ssh/authorized_keys2"
```

An alternative is to create a link from `authorized_keys` to `authorized_keys2`:

```
$ "cd $HOME/.ssh "
$ "ln -s authorized_keys authorized_keys2"
```

On the local (client) machine test the results by connecting to the remote (server) machine:

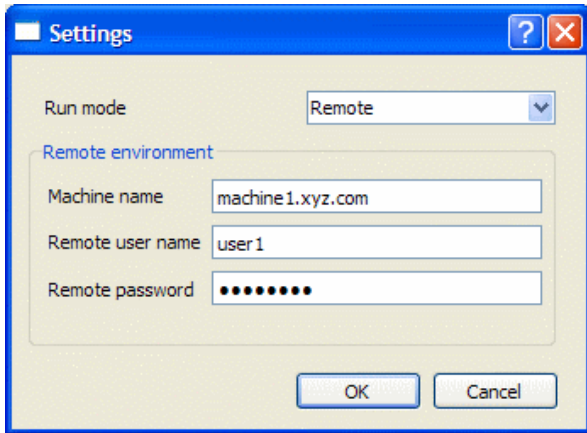
```
$ "ssh -i $HOME/.ssh/id_dsa <remote_machine_name>"
```

When you login the first time, you may be asked for confirmation yes/no. Enter `yes` to proceed. If everything is correct, you can login without any password to the remote machine.

If there is an issue, you may need to consult your system administrator.

To run Assura DRC in Remote Mode:

1. From your Layout window, click **Tools > DRC**.
2. Choose **Assura** as the *DRC Engine* and click **Settings** in the Design Rule Check dialog box.
3. Choose **Remote** for the *Run mode* in the Assura DRC Link dialog box.



- 3.
4. Enter the **Machine name** and the **Remote user name**.
5. Click **OK** in the Assura DRC Link dialog box to return to the Design Rule Check dialog box.
6. Click **Browse** to choose the *Rule file* and click **OK**.
7. Click **Run** to execute the Assura DRC.

Note

By default, an ADS-Assura DRC is executed on the entire layout. To run Assura DRC on a subsection of your layout, specify the coordinates of the desired area in the rule file.

Viewing Assura DRC Link Results

After the Assura DRC results are available, ADS generates a compact report file from the Assura error with summary files (.err and .sum files), translates the report file to the Jade format, then displays Assura DRC Link results in the *ADS DRC Results Viewer*.

