Design Rule Checker

August 2005
Notice

The information contained in this document is subject to change without notice.

Agilent Technologies makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Agilent Technologies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms that apply to this software product is available upon request from your Agilent Technologies representative.

Restricted Rights Legend

Use, duplication or disclosure by the U. S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DoD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

© Agilent Technologies, Inc. 1983-2005
395 Page Mill Road, Palo Alto, CA 94304 U.S.A.

Acknowledgments

Mentor Graphics is a trademark of Mentor Graphics Corporation in the U.S. and other countries.

Microsoft®, Windows®, MS Windows®, Windows NT®, and MS-DOS® 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.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

SystemC® is a registered trademark of Open SystemC Initiative, Inc. in the United States and other countries and is used with permission.
# Contents

## 1 DRC Quick Start
- Rule Registry File ..................................................................................................... 1-2
- Rule Directories ........................................................................................................ 1-2
- Setting Up a Quick DRC .......................................................................................... 1-3
- Setting Up a Custom DRC ....................................................................................... 1-5
- Running a DRC ........................................................................................................ 1-7
- Viewing DRC Results ............................................................................................. 1-8
- Viewing DRC Errors ............................................................................................... 1-9
- Saving a DRC Rule .................................................................................................. 1-11
- Reloading DRC Results .......................................................................................... 1-12
- Viewing DRC Examples .......................................................................................... 1-14
- Copying a DRC Example ....................................................................................... 1-14

## 2 Writing Design Rules
- Extension and Intrusion Definitions ........................................................................ 2-1
- Anatomy of a Simple DRC Rule File ........................................................................ 2-2
- Layer Management .................................................................................................. 2-3
  - Import Layers ....................................................................................................... 2-3
  - Export Layers ...................................................................................................... 2-4
  - Work Layers ....................................................................................................... 2-5
  - Rules File Layers Example ................................................................................. 2-5
- Complete DRC Example ........................................................................................... 2-6
  - Defining the Design Layers .................................................................................. 2-7
  - Defining the Error Layers .................................................................................... 2-7
  - Checking the Clearance Rules ............................................................................. 2-8
  - Selecting Polygons .............................................................................................. 2-14
  - Using Rule Conjunction .................................................................................... 2-18

## 3 DRC Layer Management Commands
- dve_import_layer() .............................................................................................. 3-2
- dve_export_layer() ............................................................................................... 3-3

## 4 Conditional Selection
- dve_drc() ............................................................................................................ 4-2
- dve_combine() ..................................................................................................... 4-4
- Edge Selection Based On Clearance ....................................................................... 4-5
  - gap() ................................................................................................................. 4-5
  - notch() .............................................................................................................. 4-6
  - single_clearance() .............................................................................................. 4-7
  - spacing() ......................................................................................................... 4-8
  - width() ............................................................................................................. 4-12
5 Polygon Selection
Polygon Selection Based on Intrinsic Properties................................................. 5-2
  poly_area() ....................................................................................................... 5-3
  poly_hole_count() .......................................................................................... 5-4
  poly_line_length() .......................................................................................... 5-7
  poly_perimeter() .............................................................................................. 5-8
Polygon Selection Based on Merge Properties..................................................... 5-9
  Convention for TOP and BOTTOM layers....................................................... 5-9
  Using edge codes............................................................................................ 5-9
  Edge Code Qualifier ....................................................................................... 5-11
  Definition of paths and anti-paths................................................................. 5-11
  poly_edge_code() ........................................................................................... 5-12
  poly_path_count() .......................................................................................... 5-15
  poly_path_length() .......................................................................................... 5-17
Polygon Selection Based on Edge Relationships................................................ 5-20
  poly_inter_layer() .......................................................................................... 5-21

6 Boolean Operations on Edges
  all_edges() ..................................................................................................... 6-2
  invert_edges() .................................................................................................. 6-3

7 Operations for Polygon Extraction from Edges
  dve_plgout() ..................................................................................................... 7-2
  dve_quadout() .................................................................................................. 7-3

8 Merge Operations on Polygons
  dve_bool_and() ............................................................................................... 8-2
  dve_bool_not() ................................................................................................. 8-3
  dve_bool_or() ................................................................................................... 8-4
  dve_merge() ..................................................................................................... 8-5
9 Sizing Operations on Polygons
   dve_oversize() ................................................................................................. 9-2
   dve_undersize ()............................................................................................. 9-3

10 Macros
   intrusion() ......................................................................................................... 10-2
   protrusion() ..................................................................................................... 10-4

11 Troubleshooting
   Layer Management Errors (101-199) ................................................................. 11-1
   Layer Management Warnings (201-299) .............................................................. 11-2
   Command Usage Errors (301-399) .................................................................... 11-2
   Command Usage Warnings (401-499) ................................................................. 11-4

Index
Chapter 1: 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 custom check using prewritten rules to ensure that a design meets manufacturing specifications. In either case, you can check all or part of the design.

Whether you run a quick check or a custom check, 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 Refer to the Customization and Configuration manual, Chapter 3, Setting Layout Options, for information on setting preferences for DRC’s memory usage and performance.

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. 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.
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_1>, <rule_file_1>,
    <rule_name_2>, <rule_file_2>,
    ........
    <rule_name_n>, <rule_file_n>
));
```

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 Project directory shown below, the Rule Selection dialog displays this list of rules when you click the Project button.

```
// 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"
));
```

Rule Directories

You can store a rule file in any directory and find the file using the Rule File Browser invoked by the Browse button in the Rule Selection dialog box. However, it is better to
use one of the four rule directories supported by the program to facilitate rule file browsing. The four rule directories for storing custom rules are: Site, User, Project and Design Kit. Buttons corresponding to these directories are available in the dialog box. You can find prewritten design rules at each level.

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: Width/Spacing to open the DRC dialog box. Use the first tab to define a basic design rule to be used.
DRC Quick Start

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.

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.

3. 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.
Setting Up a Custom DRC

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

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

To set up a custom DRC:

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.
3. Choose Tools > DRC: Custom Rules to open the Custom DRC dialog box at the Select Rule tab.

4. In the Apply to Layer(s) list select the design layer(s) you want checked.
5. Click Apply to start the process.
DRC Quick Start

- The Rules List allows the selection of a DRC rules file from one of several predefined locations. Use the Site, User, Project, or Design Kit button to see a list of rules in each of these locations. Then select a specific rules file by selecting a description of the rules in the window.

- The Rules File displays the selected rules file.

- Browse... displays the Select Rules File dialog box where you can select a rules file from other locations.

For details, see “Rule Directories” on page 1-2 and “Rule Registry File” on page 1-2.

4. Choose Site, User, Project, or Design Kit to view predefined rules. The Design Kit button will allow rules to be chosen from enabled design kits. If more than one design kit is enabled, a dialog box will be displayed, allowing you to choose which design kit to use.
5. Select a rule from the Rules List. The description of the design rules displays in the selected list. If the rule you want is a rule file, you can browse to find it or you can enter the path and rule file name in the Rule File field.

6. Click **Apply** to compile the selected rule. The DRC Message Window opens and displays a running message similar to the example:

7. Click **OK** to save the file.

---

**Running a DRC**

To run a DRC:

1. In the DRC dialog, click **Run DRC**.

2. In the Check Area, choose whether you want the program to check the entire layout or only the area that is currently visible in the Layout window. You can save time on large designs by checking only the area of concern.
3. You can accept the default Job Name or enter a different name. The default Job Name is the design name with the suffix _drc. In either case, include the suffix.
4. Click **Apply** to start the process.

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

Run DRC Job <Job Name>_drc for full design...
DRC process complete

**Viewing DRC Results**

After running a DRC check, you can view the results. See also “Reloading DRC Results” on page 1-12 and “Viewing DRC Errors” on page 1-9.

To view results:

1. In the DRC dialog, select **Load Result**.
2. In the Job Name list, select the job name you want to view.
3. Click **Apply** to view the results. A message similar to the example displays in the message window.

```
Load results <Job Name>_drc
Load results complete
```

**Viewing DRC Errors**

After running a design rule check, you can view a summary of the results and any errors found by the check.

**Note** For a quick DRC, the program automatically creates a drc layer on which error segments are displayed. For a custom DRC, the program does not create a drc layer automatically. You must create an appropriate drc layer(s) before you run the check.

To view errors:

1. In the DRC dialog, select **View Errors**.
DRC Quick Start

2. Click First.

If there are no errors, the message window displays:

No DRC error exists!

If at least one error exists, the error segment(s) in the Layout window are highlighted and the message window displays:

Error #1:
<design rule>

where <design rule> is what you defined previously in the Define Rule tab. For example:

Width of layer cond must be >= 25.00

3. Enable Auto Select and Auto Zoom, then click First again. In the Layout window, the program zooms in on the area that contains the first error and the error segments are selected so you can delete the DRC segments as you fix problems in the layout.

4. Click Next. If there is more than one error, the message window displays the next error and the program moves the zoomed display in the Layout window to the area that contains the next error.
5. Click **Summary**. The message window displays a summary similar to the example.

- **Job Name**: example_drc
- **Design Name**: example
- **Design Rule**: `<current project directory path>/verification/autorule.ael`
- **Total Number of Errors**: 1

**Note** If you prefer, you can view the summary before you view any errors.

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.

### Saving a DRC Rule

You can save the rule that is created when you run a Quick DRC. You can define a name for the rule, a name for the AEL file, and where you want the file stored before you define the rule or you can save the rule after you create it. When you save a Design Rule, the program automatically updates the rule registry file to include the new rule (see "Rule Registry File" on page 1-2).

To save a design rule:

1. After defining a rule, click **Save As** in the Define Rule tab.
DRC Quick Start

To reload DRC results:

1. In the DRC dialog, select **Load Result**.
2. Select the **Job Name** for the results you want to view.
3. Select Apply to display the results.
DRC Quick Start

Viewing DRC Examples

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. However, the basic path is the same.

To view an example:

1. In the Advanced Design System Main window, select the Examples button on the toolbar.
2. In the File Browser, select the directory path to the example designs; examples/MW_Ckts/drc_via_prj/networks/pwramp.dsn

Copying a DRC Example

The files in the examples directory are read-only, so you must copy them to your directory before you can run the examples.

To copy a DRC example:

1. In the Main window, select File > Copy Project.
2. In the Copy Project dialog box, select Example Directory.
3. Select Browse.
4. In the Copy From File Browse dialog box, double-click the project directory.
5. From the list of files in the selected project directory, select a project.
6. Click OK.
7. In the Copy Project dialog box, click Startup Directory as the To Project.
8. Click **Browse**.

9. In the Copy To File Browse dialog box, select the project directory, then click **OK**.

10. In the Copy Project dialog box, click at the end of the path displayed as the To Project and enter a file name (including suffix) for the copied project.

11. Confirm that Copy Project Hierarchy is selected.

12. Click **OK** to copy the project.
Chapter 2: Writing Design Rules

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

- 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 chapters.

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
dcl backVia = dve_import_layer(20);

// declare output layer
dcl lyrDRCError = dve_export_layer(101);

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

lyrDRCError += 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 error101. Each violation has an associated error message: width less than 3.0.

The AEL variable lyrCond references an import layer and the AEL variable lyrError101 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:

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

or

```c
decl lyrCond = dve_import_layer (1);
```
Writing Design Rules

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).

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 lyrError101 = dve_export_layer ("error101");
```

or

```
decl lyrError101 = 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 error101.

dcl lyrCond = dve_import_layer ("cond");
dcl lyrCond2 = dve_import_layer ("cond2");
dcl lyrError101 = dve_export_layer ("error101");
dcl lyrPolyOverlap = NULL;
lyrPolyOverlap = dve_bool_and (lyrCond, lyrCond2);
lyrError101 += dve_drc (all_edges (lyrPolyOverlap),
   "Conductive metal cond overlaps cond2");
Writing Design 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_prj directory of the program’s examples directory. For information on accessing the examples directory, see “Viewing DRC Examples” on page 1-14.

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” on page 2-7 and “Defining the Error Layers” on page 2-7.

Table 2-1 shows the layer definitions for the process used in this example.

<table>
<thead>
<tr>
<th>Mask Level</th>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment Key</td>
<td>13</td>
<td>Defines fields in which alignment artifacts will be etched.</td>
</tr>
<tr>
<td>N+ Implant</td>
<td>2</td>
<td>Mask during alignment artifact etch, Implant mask for N+ regions.</td>
</tr>
<tr>
<td>D- Implant</td>
<td>1</td>
<td>Implant mask for DFET channels, Half DFET Diodes, D-Resistors.</td>
</tr>
<tr>
<td>NiCr</td>
<td>3</td>
<td>Liftoff layer for NiCr Resistors</td>
</tr>
<tr>
<td>Ohmic</td>
<td>5</td>
<td>Liftoff layer for ohmic contact on GaAs devices. Ohmic Metal may NOT be used for interconnect.</td>
</tr>
<tr>
<td>Isolation Implant</td>
<td>6</td>
<td>Implant mask for Isolation Implant</td>
</tr>
<tr>
<td>Gate Metal</td>
<td>7</td>
<td>Liftoff layer Schottky Gate/Anode contact on GaAs devices. Gate Metal may NOT be used for interconnect</td>
</tr>
<tr>
<td>Metal 0</td>
<td>9</td>
<td>Liftoff layer for Metal 0</td>
</tr>
<tr>
<td>MIM</td>
<td>23</td>
<td>Liftoff layer for MIM metal</td>
</tr>
<tr>
<td>Via 1</td>
<td>14</td>
<td>First via etch layer</td>
</tr>
<tr>
<td>Metal 1</td>
<td>15</td>
<td>First plated Au metal layer. Labels are done in this layer</td>
</tr>
<tr>
<td>Air Bridge Post</td>
<td>10</td>
<td>Support Posts for Air Bridge and Via to Metal1</td>
</tr>
<tr>
<td>Air Bridge</td>
<td>11</td>
<td>Second plated Au metal layer</td>
</tr>
<tr>
<td>Passivation Via</td>
<td>12</td>
<td>Opens vias over bond pads and saw streets</td>
</tr>
</tbody>
</table>
Defining the Design Layers

The rule section declares these imported design layers:

```c
// 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.

```c
// 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
```
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" on page 2-8
- "spacing" on page 2-9
- "external" on page 2-10
- "contains" on page 2-11
- "nests" on page 2-12
- "internal" on page 2-13

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.

Note: DRC error message strings: ( ) and _ are supported, but ‘ ’ are not supported.
Width rules for the substrate via are written as follows:

```c
// 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.

```
Table 2-2. Substrates Via Design Rules

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Minimum (um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Coded Substrate Via Feature, Square (layer 20)</td>
<td>30</td>
</tr>
<tr>
<td>B</td>
<td>Substrate Via Target (layer 7)</td>
<td>120</td>
</tr>
</tbody>
</table>
```

```

**Table 2-3. Substrate Via Design Rule**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Minimum (um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Substrate Via (layer 20) to Via (20), Edge to Edge</td>
<td>150</td>
</tr>
</tbody>
</table>
```

```
//
// 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");
```
Writing Design Rules

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” on page 2-18.

eexternal

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

Table 2-4. Substrate Via Design Rule

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Minimum(um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Substrate Via (layer 20) to Active Device Edge (layer 6)</td>
<td>90</td>
</tr>
</tbody>
</table>

//
// 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.

```
// Rule E - Metal 0 Inclusion in Gate Metal min is 1 um

gateMetalError += dve_drc(contains(gateMetal, metal0) < 90,
    "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.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Minimum(um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Metal 0 (layer 9) Inclusion in Gate Metal (layer 7)</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 2-5. Substrate Via Design Rule
Writing Design Rules

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,

```
Table 2-6. NiCr Thin Film Resistors Design Rule

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Minimum(um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Metal 0 (layer 9) Extension from NiCr (layer 3)</td>
<td>0.5</td>
</tr>
</tbody>
</table>
```

2-12 Complete DRC Example
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.
Writing Design Rules

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 Chapter 4, Conditional Selection.

poly_path_length

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

Table 2-7. NiCr Thin Film Resistors Design Rule

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Minimum(um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>NiCr (layer 3) Intrusion into Metal0 (layer 9)</td>
<td>2.5</td>
</tr>
</tbody>
</table>

//
// 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");
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” on page 5-9.

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;
```
Writing Design Rules

//
// 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 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.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Minimum(um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Metal 0 (layer 9) Inclusion in Gate Metal (layer 7)</td>
<td>1.0</td>
</tr>
</tbody>
</table>

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:

```c
// // Substrate Via Spacing Design Rules // Rule E - Metal 0 Inclusion in Gate Metal min is 1 um //

// declare some work layers
dcl 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
Writing Design Rules

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()” on page 5-21.

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 to use rule conjunction to check Gate Metal manufacturing rules.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Minimum (um)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>Gate Metal (layer 7) spacing when</td>
<td></td>
</tr>
<tr>
<td></td>
<td>width &lt; 1.5</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>1.5 &lt;= width &lt; 2.0</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>2.0 &lt;= width &lt; 3.0</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>3.0 &lt;= width</td>
<td>3.0</td>
</tr>
</tbody>
</table>

// 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
//    fateMet20Lt
// 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
// 1. select the edges with width >= 2.0 from gateMetal, save in
//    gatMet20Ge
// 2. select the edges with width < 3.0 from gatMet20Ge, save in
//    gatMet30Lt
// 3. select the edges with spacing error by checking the distance
//    between gateMetal and gatMet30Lt

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
// 1. select the edges with width >= 3.0 from gateMetal, save in
//    gatMet30Ge
// 2. select the edges with spacing error by checking the distance
//    between gateMetal and gatMet30Ge

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 .ael. For details, see “Saving a
DRC Rule” on page 1-11.
Writing Design Rules
Chapter 3: DRC Layer Management

Commands

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

- “dve_import_layer()” on page 3-2
- “dve_export_layer()” on page 3-3
DRC Layer Management Commands

**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()” on page 3-3

**Syntax**

```
inputLayer = dve_import_layer (layerId);
```

where:

- `layerId` is the string layer name or integer layer number of an existing design layer

**Example**

```
decl lyrCond = dve_import_layer ("cond");
decl lyrError101 = dve_export_layer ("error101");

lyrError101 += dve_drc (width (lyrCond) < 4.0,
  "Metal width less than 4.0");
```
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()" on page 3-2

Syntax

\[
\text{exportLayer} = \text{dve\_export\_layer}\ (\text{layerId});
\]

where:

\[
\text{layerId} \quad \text{is the string layer name or integer layer number of an existing design layer}
\]

Example

// Import layers
\text{decl} \ lyrCond = \text{dve\_import\_layer}\ ("cond");
\text{decl} \ lyrCond2 = \text{dve\_import\_layer}\ ("cond2");

// Export layers
\text{decl} \ lyrError101 = \text{dve\_export\_layer}\ ("error101");
\text{decl} \ lyrError102 = \text{dve\_export\_layer}\ ("error102");

// Work layer
\text{decl} \ lyrOverlap = \text{NULL};

// Export DRC error directly to an export layer
lyrError101 += \text{dve\_drc}\ (\text{width} (\text{lyrCond}) < 4.0,
               "Metal width less than 4.0");
lyrOverlap = \text{dve\_bool\_and}\ (\text{lyrCond}, \text{lyrCond2});
lyrError102 += \text{dve\_drc}\ (\text{all\_edges} (\text{lyrOverlap}),
               "Metal layers overlap");
DRC Layer Management Commands
Chapter 4: Conditional Selection

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

• “dve_drc()” on page 4-2
• “dve_combine()” on page 4-4
• “Edge Selection Based On Clearance” on page 4-5
  • “gap()” on page 4-6
  • “notch()” on page 4-7
  • “single_clearance()” on page 4-8
• “spacing()” on page 4-10
• “width()” on page 4-12
• “contains()” on page 4-14
• “double_clearance()” on page 4-16
• “external()” on page 4-18
• “internal()” on page 4-20
• “nests()” on page 4-22
• “Edge Qualifiers” on page 4-24
• “Edge Selection Based on Corners” on page 4-36
  • “corner_edges()” on page 4-37
• “Edge Selection Based on Grid” on page 4-40
  • “off_grid()” on page 4-41
• “Edge Compensation” on page 4-42
  • “compensate()” on page 4-43
  • “dve_segsize()” on page 4-48
Conditional Selection

**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:

- `drc_expression` is an AEL expression in the format:
  `drc_subfunction ((parameter, ...)) [operator rValue])`

- `drc_subfunction` 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:
  
  - **“Edge Selection Based On Clearance” on page 4-5** (output layer contains polygons with selected edges) selection functions are separated by number of layers:
    1 Layer check: gap, notch, single_clearance, spacing, width
    2 Layer check: contains, double_clearance, external, internal, nests,
  
  - **“Edge Selection Based on Corners” on page 4-36** selection functions include: corner_edges
  
  - **“Edge Selection Based on Grid” on page 4-40** selection functions include: off_grid
  
  - **“Edge Compensation” on page 4-42** selection functions include: compensate, dve_segsize
  
  - **“Polygon Selection Based on Intrinsic Properties” on page 5-2** (output layer contains polygons) selection functions include: poly_area, poly_hole_count, poly_line_length, poly_perimeter
  
  - **“Polygon Selection Based on Merge Properties” on page 5-9** (output layer contains polygons) selection functions include: poly_edge_code, poly_path_count, poly_path_length
  
  - **“Polygon Selection Based on Edge Relationships” on page 5-20** (output layer contains polygons) selection functions include: poly_inter_layer
Example

decl lyrCond = dve_import_layer ("cond");
decl lyrError101 = dve_export_layer ("error101");

lyrError101 += dve_drc (width (lyrCond) < 3.0,
    "Width of conductive metal < 3.0");
Conditional Selection

**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:

- *inLayer1, inLayer2, inLayerN* A polygon layer

**Example**

dcl lyrCond = dve_import_layer("cond");
dcl lyrCond2 = dve_import_layer("cond2");
dcl lyrDiel = dve_import_layer("diel");

dcl lyrError101 = dve_export_layer("error101");

// collect layers cond and cond2 into the same layer so that shapes // on these layers are checked together
dcl lyrPolyOverlap = dve_combine(lyrCond ,lyrCond2);

// apply clearance rule
lyrError101 += dve_drc(double_clearance(lyrDiel,lyrPolyOverlap) < 5,
"Conductive metal overlaps", DVE_RN_POLARITY_FROM, DVE_RV_INSIDE,
DVE_RN_POLARITY_TO, DVE_RV_OUTSIDE);

![Diagram of layers cond and cond2](image-url)
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:
- “gap()” on page 4-6
- “notch()” on page 4-7
- “single_clearance()” on page 4-8
- “spacing()” on page 4-10
- “width()” on page 4-12

2 Layer Check:
- “contains()” on page 4-14
- “double_clearance()” on page 4-16
- “external()” on page 4-18
- “internal()” on page 4-20
- “nests()” on page 4-22
Conditional Selection

**gap()**

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

See also: “dve_drc()” on page 4-2

**Syntax**

\[
dve\_drc \left( \text{gap (inLayer)} \ \text{operator} \ \text{distance} \ [, \ \text{msgString}] \ [, \ \text{qualifierName}, \ \text{qualifierValue}... \] \right);
\]

where:

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

**Edge Qualifiers**

“DVE_RN_EDGE_ANGLES” on page 4-24

“DVE_RN_ANGLE_TOLERANCE” on page 4-24

“DVE_RN_SEPARATE” on page 4-25

“DVE_RN_TOUCH” on page 4-28

“DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO” on page 4-26

“DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM” on page 4-30

**Example**

```c
decl lyrCond = dve_import_layer ("cond");
decl lyrError101 = dve_export_layer ("error101");

// Check between outside edges of polygons on same layer
lyrError101 += dve_drc (gap (lyrCond) < 4.0, "Outside edges < 4.0");
```

---

4-6 Edge Selection Based On Clearance
notch()

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

See also: “dve_drc()” on page 4-2

Syntax

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

where:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inLayer</td>
<td>A polygon layer</td>
</tr>
<tr>
<td>operator</td>
<td>&lt;  Less than</td>
</tr>
<tr>
<td></td>
<td>&lt;= Less than or equal to</td>
</tr>
<tr>
<td></td>
<td>== Equal to</td>
</tr>
<tr>
<td></td>
<td>&gt;  Greater than</td>
</tr>
<tr>
<td></td>
<td>&gt;= Greater than or equal to</td>
</tr>
<tr>
<td>distance</td>
<td>A distance value in layout units</td>
</tr>
<tr>
<td>msgString</td>
<td>A string value that will be attached to the selected error segments</td>
</tr>
<tr>
<td>qualifierName, qualifierValue</td>
<td>A name, value pair that qualifies the selection</td>
</tr>
</tbody>
</table>

**Edge Qualifiers**

“DVE_RN_EDGE ANGLES” on page 4-24

“DVE_RN_ANGLE_TOLERANCE” on page 4-24

“DVE_RN_SEPARATE” on page 4-25

“DVE_RN_TOUCH” on page 4-28

“DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO” on page 4-26

“DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM” on page 4-30

**Example**

dcl lyrCond2 = dve_import_layer ("cond2");
dcl lyrError101 = dve_export_layer ("error101");

lyrError101 += dve_drc (notch (lyrCond2) < 15.0,
    "Outside edges same polygon < 15.0");
Conditional Selection

**single_clearance()**

Measures the distance between edges of a single polygon.

See also: “dve_drc()” on page 4-2

**Syntax**

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

where:

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

**Edge Qualifiers**

“DVE_RN_POLARITY, DVE_RN_POLARITY_FROM, DVE_RN_POLARITY_TO” on page 4-24

“DVE_RN_STRUCTURE” on page 4-25

“DVE_RN_EDGE_ANGLES” on page 4-24

“DVE_RN_ANGLE_TOLERANCE” on page 4-24

“DVE_RN_SEPARATE” on page 4-25

“DVE_RN_TOUCH” on page 4-28

“DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO” on page 4-26

“DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM” on page 4-30

**Example**

```
decl lyrCond = dve_import_layer ("cond");
```
decl lyrError101 = dve_export_layer ("error101");

lyrError101 += dve_drc (single_clearance (lyrCond) < 3.0, 
"Parallel clearance < 3.0",
  DVE_RN_POLARITY, DVE_RV_OUTSIDE,
  DVE_RN_TEMPLATE, DVE_RV_OPPOSITE,
  DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL,
  DVE_RN_ANGLE_TOLERANCE, 1.2);
Conditional Selection

**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()” on page 4-2

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

where:

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

**Edge Qualifiers**

“DVE_RN_EDGE_ANGLES” on page 4-24
“DVE_RN_ANGLE_TOLERANCE” on page 4-24
“DVE_RN_SEPARATE” on page 4-25
“DVE_RN_TOUCH” on page 4-28
“DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO” on page 4-26
“DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM” on page 4-30

**Example**

```plaintext
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrError101 = dve_export_layer ("error101");
lyrError101 += dve_drc (spacing (lyrCond2) < 15.0,
```

---

4-10 Edge Selection Based On Clearance
"Gap and notch spacing < 15.0";
Conditional Selection

**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()” on page 4-2

**Syntax**

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

where:

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

**Edge Qualifiers**

“DVE_RN_EDGE_ANGLES” on page 4-24
“DVE_RN_ANGLE_TOLERANCE” on page 4-24
“DVE_RN_SEPARATE” on page 4-25
“DVE_RN_TOUCH” on page 4-28
“DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO” on page 4-26
“DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM” on page 4-30

**Example**

```plaintext
decl lyrCond = dve_import_layer ("cond");
decl lyrError101 = dve_export_layer ("error101");
lyrError101 += dve_drc (width (lyrCond) < 3.0,
```
"Width of metal layer < 3.0";
Conditional Selection

contains()

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

Syntax

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

where:

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

Edge Qualifiers

- “DVE_RN_EDGE_ANGLES” on page 4-24
- “DVE_RN_ANGLE_TOLERANCE” on page 4-24
- “DVE_RN_SEPARATE” on page 4-25
- “DVE_RN_TOUCH” on page 4-28
- “DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO” on page 4-26
- “DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM” on page 4-30

Example

dcl lyrCond = dve_import_layer ("cond");
dcl lyrCond2 = dve_import_layer ("cond2");
dcl lyrError101 = dve_export_layer ("error101");
lyrError101 += dve_drc (contains (lyrCond, lyrCond2) < 3.0, "Enclosure clearance < 3.0");
Conditional Selection

**double_clearance()**

Measures the distance between edges of polygons on different layers.

See also: "dve_drc()" on page 4-2

**Syntax**

dve_drc (double_clearance (inLayer1, inLayer2) operator distance 

[, msgString] [,qualifierName, qualifierValue...]);

where:

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

**Edge Qualifier**

"DVE_RN_POLARITY, DVE_RN_POLARITY_FROM, DVE_RN_POLARITY_TO" on page 4-24

"DVE_RN_EDGE_ANGLES" on page 4-24

"DVE_RN_ANGLE_TOLERANCE" on page 4-24

"DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO" on page 4-26

"DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM" on page 4-30

**Example**
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrError101 = dve_export_layer ("error101");

lyrError101 += dve_drc (double_clearance (lyrCond, lyrCond2) < 3.0,
    "Metal layers run parallel and close",
    DVE_RN_POLARITY, DVE_RV_OUTSIDE,
    DVE_RN_TEMPLATE, DVE_RV_OPPOSITE,
    DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL,
    DVE_RN_ANGLE_TOLERANCE, 1.2);
Conditional Selection

**external()**

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

See also: “dve_drc()” on page 4-2

**Syntax**

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

where:

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

**Edge Qualifiers**

“DVE_RN_EDGE_ANGLES” on page 4-24

“DVE_RN_ANGLE_TOLERANCE” on page 4-24

“DVE_RN_SEPARATE” on page 4-25

“DVE_RN_TOUCH” on page 4-28

“DVE_RN_SLOPE, DVE_RN_SLOPE_FROM, DVE_RN_SLOPE_TO” on page 4-26

“DVE_RN_TEMPLATE, DVE_RN_TEMPLATE_TO, DVE_RN_TEMPLATE_FROM” on page 4-30

**Example**

decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrError101 = dve_export_layer ("error101"神通2);
lyrError101 += dve_drc (external (lyrCond, lyrCond2) < 4.0,
  "Outside edges of metal layers < 4.0",
  DVE_RN_EDGEANGLES, DVE_RV_PARALLEL);
Conditional Selection

**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()" on page 4-2

**Syntax**

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

where:

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

**Edge Qualifiers**

"DVE_RN_EDGEANGLES" on page 4-24
"DVE_RN ANGLETOLERANCE" on page 4-24
"DVE_RN SEPARATE" on page 4-25
"DVE_RN TOUCH" on page 4-28
"DVE_RN SLOPE, DVE_RN SLOPE_FROM, DVE_RN SLOPE_TO" on page 4-26
"DVE_RN TEMPLATE, DVE_RN TEMPLATE_TO, DVE_RN TEMPLATE_FROM" on page 4-30

**Example**

dcl lyrCond = dve_import_layer ("cond");
dcl lyrCond2 = dve_import_layer ("cond2");
dcl lyrError101 = dve_export_layer ("error101");
lyrError101 += dve_drc (internal (lyrCond, lyrCond2) < 4.0,
"Inside edges < 4.0");
Conditional Selection

\textbf{nests()}

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

See also: "dve_drc()" on page 4-2

\textbf{Syntax}

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

where:

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

\textbf{Edge Qualifiers}

“DVE_RN_EDGEANGLES” on page 4-24

“DVE_RN_Angle_Tolerance” on page 4-24

“DVE_RN_Separate” on page 4-25

“DVE_RN_Touch” on page 4-28

“DVE_RN_Slope, DVE_RN_Slope_From, DVE_RN_Slope_To” on page 4-26

“DVE_RN_Template, DVE_RN_Template_To, DVE_RN_Template_From” on page 4-30

\textbf{Example}

decl lyrCond = dve_import_layer("cond");
decl lyrCond2 = dve_import_layer("cond2");
decl lyrError101 = dve_export_layer("error101");
decl lyrError102 = dve_export_layer ("error102");

lyrError101 += dve_drc (nests (lyrCond, lyrCond2) < 4.0,
    "Clearance from contained to containing layers < 4.0");
lyrError102 += dve_drc (nests (lyrCond2, lyrCond) < 4.0,
    "Clearance from contained to containing layers < 4.0");
Conditional Selection

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_EDGE_ANGLES**

Qualifier Resource Value:

- `DVE_RV_PARALLEL` Select only parallel edges
- `DVE_RV_NOT_PARALLEL` Select only non-parallel edges
- `DVE_RV_PERPENDICULAR` Select only perpendicular edges
- `DVE_RV_NOT_PERPENDICULAR` Select only non-perpendicular edges
- `DVE_RV_ANY_ANGLE` (default) 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:

`<real value>`  Edge angle tolerance in degrees

This qualifier can only be used in conjunction with RUL_RN_EDGE_ANGLES.

For example:

```plaintext
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.

**Note**  Any custom DRC rules need to be updated to the correctly spelled version of this command, TOLERANCE. TOLLERANCE, with two L's is no longer supported.

**DVE_RN_POLARITY, DVE_RN_POLARITY_FROM, DVE_RN_POLARITY_TO**

4-24  Edge Selection Based On Clearance
Qualifier Resource Value:

\[ DVE_{-}RV_{-}INSIDE \]
Direct search toward inside of polygon

\[ DVE_{-}RV_{-}OUTSIDE \]
(default) Direct search toward outside of polygon

**DVE_RN_STRUCTURE**

Qualifier Resource Value:

\[ DVE_{-}RV_{-}ANY_{-}POLYGON \]  (default) Test applies to any edge

\[ DVE_{-}RV_{-}SAME_{-}POLYGON \]
Test applies only between edge of same polygon

\[ DVE_{-}RV_{-}DIFF_{-}POLYGON \]
Test applies only between edge of different polygons

**DVE_RN_SEPARATE**

Determines how two adjacent edges are checked.

Qualifier Resource Value:

\[ DVE_{-}RV_{-}SEPARATE \]
applies only to non-intersecting edges

\[ DVE_{-}RV_{-}NOT_{-}SEPARATE \]
applies only to intersecting edges

\[ DVE_{-}RV_{-}ANY_{-}SEPARATE \]
applies to edges regardless of whether they intersect or not

\[ DVE_{-}RV_{-}PERP_{-}SEPARATE \]
applies only if two adjacent edges are not perpendicular

\[ DVE_{-}RV_{-}JOIN_{-}SEPARATE \]
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:
Conditional Selection

```
lyrError101 = 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.
lyrError101 = dve_drc(width(lyrEdges) < 100.0, DVE_RN_SEPARATE, DVE_RV_ANY_SEPARATE);
```

2. See the command “corner_edges()” on page 4-37 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:**

- **DVE_RV_VERTICAL** Select only vertical edges
- **DVE_RV_HORIZONTAL** Select only horizontal edges
- **DVE_RV_ORTHOGONAL** Select only vertical and horizontal edges
- **DVE_RV_DIAGONAL** Select only diagonal edges
- **DVE_RV_OCTAGONAL** Select only vertical, horizontal and diagonal edges
DVE_RV_OTHER  Select only non-octagonal edges
DVE_RV_ALL_SLOPES (default) Select edges at any slope
Conditional Selection

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

```
lyrError101= dve_drc(gap(lyrCond) < 35.0, DVE_RN_SLOPE, DVE_RV_VERTICAL);
```

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

```
lyrError101= dve_drc(gap(lyrCond) < 35.0, DVE_RN_SLOPE, DVE_RV_DIAGONAL);
```

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

```
lyrError101= dve_drc(gap(lyrCond) < 35.0, DVE_RN_SLOPE_FROM, DVE_RV_ORTHOGONAL, DVE_RN_SLOPE_TO, DVE_RV_DIAGONAL);
```

**DVE_RN_TOUCH**

Qualifier Resource Value:

---

4-28 Edge Selection Based On Clearance
Examples
1. Consider two polygons on layer lyrCond and lyrCond2 which touch externally and an external rule:

```
lyrError101 = 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:

```
lyrError101 = 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:

```
lyrError101 = dve_drc(nests(lyrCond2, lyrCond) < 30.0, "d min is 30", DVE_RN_EDGE_ANGLES, DVE_RV_PARALLEL);
```

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.
Conditional Selection

\[ \text{lyrError101} = \text{dve_drc}(\text{nests(lyrCond2,lyrCond)} < 30.0, \text{"d min is 30"}, \text{DVE_RN_EDGE_ANGLES}, \text{DVE_RV_PARALLEL}, \text{DVE_RN_TOUCH}, \text{DVE_RV_OVERLAP}); \]

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

Similarly,

\[ \text{lyrError101} = \text{dve_drc}(\text{external(lyrCond2,lyrCond)} < 30.0, \text{"d min is 30"}, \text{DVE_RN_EDGE_ANGLES}, \text{DVE_RV_PARALLEL}, \text{DVE_RN_TOUCH}, \text{DVE_RV_OVERLAP}); \]

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.
Conditional Selection

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:
Conditional Selection

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” on page 4-30 for the definition of FROM and TO and refer to “Specifying a Template” on page 4-32 for the definition of concave and convex corners.

Qualifier Resource Values:

<table>
<thead>
<tr>
<th>DVE_RV_ROUND</th>
<th>DVE_RV_ARC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVE_RV_ROUND_ARC</td>
<td>DVE_RV_ARC_OPPOSITE</td>
</tr>
<tr>
<td>DVE_RV_ROUND_OPPOSITE</td>
<td>DVE_RV_ARC_ROUND</td>
</tr>
<tr>
<td>DVE_RV_ROUND_SQUARE</td>
<td>DVE_RV_ARC_SQUARE</td>
</tr>
<tr>
<td>DVE_RV_OPPOSITE</td>
<td>DVE_RV_SQUARE</td>
</tr>
<tr>
<td>DVE_RV_OPPOSITE_ARC</td>
<td>DVE_RV_SQUARE_ARC</td>
</tr>
<tr>
<td>DVE_RV_OPPOSITE_ROUND</td>
<td>DVE_RV_SQUARE_OPPOSITE</td>
</tr>
<tr>
<td>DVE_RV_OPPOSITE_SQUARE</td>
<td>DVE_RV_SQUARE_ROUND</td>
</tr>
<tr>
<td>DVE_RV_BOTHSIDES</td>
<td></td>
</tr>
</tbody>
</table>
Conditional Selection

**Edge Selection Based on Corners**

Edge Selection Based on Corners selection function includes: “corner_edges()” on page 4-37.
corner_edges()
Generates error segments around corners of specified inside angles.
See also: “dve_drc()” on page 4-2

Syntax

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

where:

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

Example 1

dcl lyrCond = dve_import_layer ("cond");
dcl lyrError101 = dve_export_layer ("error101");
dcl lyrEdgesCvex = NULL;
dcl lyrEdgesStub = NULL;
dcl 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_RVSAME_POLYGON);
lyrStub = dve_quadout (lyrEdgesStub);
lyrError101 += dve_drc (all_edges (lyrStub), "Stub");
Conditional Selection

Example 2
Consider some geometry with a chamfer corner and a rule to check the width of 2.0:

```
1.5
```

This rule fails to detect the error because the default template for width() is DVE_RV_OPPPOSITE. 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** that 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_OPPPOSITE, DVE_RN_TEMPLATE_TO, DVE_RV_ARC_OPPPOSITE, DVE_RN_SEPARATE, DVE_RV_SEPARATE);
```
Note the use of the qualifier DVE_RV_SEPARATE to apply the test only to non-intersecting edges.
Edge Selection Based on Grid

Edge Selection Based on Grid selection function includes: “off_grid()” on page 4-41.
**off_grid()**
Flags edges whose end points fall off a specified grid.

See also: "dve_drc()" on page 4-2

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

where:

- \textit{inLayer} A polygon layer
- \textit{grid} A specified grid
- \textit{msgString} A string value that will be attached to the selected error segments

**Example**

dcl lyrCond = dve_import_layer ("cond");
dcl lyrError101 = dve_export_layer ("error101");

lyrError101 += dve_drc (off_grid (lyrCond, 0.5),
  "Conductive metal is off grid");
Conditional Selection

Edge Compensation

Edge Compensation selection function includes: “compensate()” on page 4-43, and “dve_segsizes()” on page 4-48.
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()” on page 7-2, “dve_quadout()” on page 7-3

Syntax

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

where:

- `edgeLayerIn`, `edgeLayerOut` An edge layer
- `distance` A real value

Compensate Template Qualifier

**DVE_RN_COMP_TEMPLATE**

Qualifier Resource Value

- `DVE_RV_CHAMFER` Compensate using an angle from the orthogonal
- `DVE_RV_ALIGN` (default) Compensate using an alignment to the adjacent edge
- `DVE_RV_BISECT` Compensate where the angle is bisected at the corner
- `DVE_RV_OPPOSITE` 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
Conditional Selection

```
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);
lyrError101  = dve_drc(all_edges(lyrPolyComp), "positive compensation");
```

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

```
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:

```
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
Conditional Selection

```c
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:

```c
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:

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

**Example**

```c
decl lyrCond = dve_import_layer ("cond");
decl lyrError101 = dve_export_layer ("error101");
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),
```

4-46 Edge Compensation
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
lyrError101 += dve_drc (gap (lyrPolyOversize) < 4.0, "Gap clearance < 4.0");
Conditional Selection

dveSegsize()

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

See also: “compensate()” on page 4-43.

Syntax

dveSegsize(errorEdgeLayer, distance);

where

errorEdgeLayer A layer with error segments
distance 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:

```plaintext
decl lyrMetal = dve_import_layer("Metal1");
decl lyrError101 = dve_export_layer("Error101");

lyrError101= 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:

```plaintext
decl lyrEdges, lyrEdgesCmp, lyrPolyCmp;
lyrEdges = dve_drc(single_clearance(lyrMetal ) < 20.0, ...);
lyrEdgesCmp = dve_drc(compensate(lyrEdges, 4));
lyrPolyCmp = dve_quadout(lyrEdgesCmp);
lyrError101 += dve_drc(all_edges(lyrPolyCmp), "clearance < 20");
```

4-48 Edge Compensation
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:

```csharp
// Define layers and operations
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);
lyrError101 += 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:

```csharp
// Define layers and operations
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 );
lyrError101 += dve_drc(all_edges(lyrPolyCmp), "clearance < 20");
```
Conditional Selection

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

```c
lyrEdgesSized = dve_segsiz(lyrEdges, 500);
```
Chapter 5: Polygon Selection

The chapter issues related to polygon selection. The topics include:

• “Polygon Selection Based on Intrinsic Properties” on page 5-2
• “Polygon Selection Based on Merge Properties” on page 5-9
• “Polygon Selection Based on Edge Relationships” on page 5-20
Polygon Selection

**Polygon Selection Based on Intrinsic Properties**

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

- “poly_area()” on page 5-3
- “poly_hole_count()” on page 5-4
- “poly_line_length()” on page 5-7
- “poly_perimeter()” on page 5-8.
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()” on page 4-2

Syntax

dve_drc (poly_area (inLayer) operator value);

where:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inLayer</td>
<td>A polygon layer</td>
</tr>
<tr>
<td>operator</td>
<td>Less than, Less than or equal to, Equal to, Greater than, Greater than or equal to</td>
</tr>
<tr>
<td>value</td>
<td>A real value in layout units</td>
</tr>
</tbody>
</table>

Example

decl lyrCond = dve_import_layer("cond");
decl lyrError101 = dve_export_layer("error101");
decl lyrPoly=NULL;

lyrPoly = dve_drc(poly_area(lyrCond) < 100);

lyrError101 += dve_drc(all_edges(lyrPoly), "Polygon area < 100");

area = 25  area = 75  area = 110
Polygon Selection

poly_hole_count()

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

See also: “dve_drc()” on page 4-2

Syntax

dve_drc (poly_hole_count (inLayer) operator numHoles);

where:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>

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.
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.

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

Polygons are extracted with the quadout command.

```plaintext
lyrPolyCmp = dve_quadout(lyrEdges);
```
Polygon Selection

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.

```c
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_EDGEANGLES, DVE_RV_PARALLEL,
DVE_RN_POLARITY, DVE_RV_INSIDE, DVE_RN_TEMPLATE, DVE_RV_OPPOSITE,
DVE_RN_STRUCTURE, DVE_RVSAME_POLYGON);
```
**poly_line_length()**

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

See also: “dve_drc()” on page 4-2

**Syntax**

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

where:

- **inLayer** A polygon layer
- **operator**
  - `<` Less than
  - `<=` Less than or equal to
  - `==` Equal to
  - `>` Greater than
  - `>=` Greater than or equal to
- **value** A real value in layout units
- **qualifierName, qualifierValue** 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

**Example**

```plaintext
decl lyrCond = dve_import_layer ("cond");
decl lyrError101 = dve_export_layer ("error101");

dec1 lyrPoly = NULL;

lyrPoly = dve_drc (poly_line_length (lyrCond) <= 10.0,
                   DVE_RN_LINE_LENGTH, DVE_RV_MAX_LINE);

lyrError101 += dve_drc (all_edges (lyrPoly), "Polygon length < 10.0");
```
Polygon Selection

poly_perimeter()

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

See also: “dve_drc()” on page 4-2

Syntax

dve_drc (poly_perimeter (inLayer) operator distance);

where:

<table>
<thead>
<tr>
<th>inLayer</th>
<th>A polygon layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator</td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>

| distance        | A real value in layout units |

Example

decl lyrCond = dve_import_layer ("cond");
decl lyrError101 = dve_export_layer ("error101");
decl lyrPoly = NULL;

lyrPoly = dve_drc (poly_perimeter (lyrCond) < 20.0);

lyrError101 += dve_drc (all_edges (lyrPoly), "Polygon perimeter < 20.0");
Polygon Selection Based on Merge Properties

Polygon Selection Based on Merge Properties selection functions include: "poly_edge_code()" on page 5-12, "poly_path_count()" on page 5-15, and "poly_path_length()" on page 5-17.

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 '*'):
Polygon Selection

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:

- **DVE_RV_TOP** (default) Select edges on top that are outside bottom
- **DVE_RV_BOT** Select edges on bottom that are outside top
- **DVE_RV_TIB** Select edges on top that are inside bottom
- **DVE_RV_BIT** Select edges on bottom that are inside top
- **DVE_RV_INT** Select edges on top and bottom that are butting internally
- **DVE_RV_EXT** 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.
Polygon Selection

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()” on page 4-2

Syntax

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

where:

- **inLayer**: A polygon layer produced by a merge operation between two layers
- **qualifierName, qualifierValue**: A name, value pair that qualifies the selection

**Selection Qualifier: DVE_RN_EDGE_SELECT**

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Resource Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVE_RV_ACCEPT_ANY</td>
<td>Select polygon if any path codes are found</td>
</tr>
<tr>
<td>DVE_RV_ACCEPT_ALL</td>
<td>(default) Select polygon if all path codes are found</td>
</tr>
<tr>
<td>DVE_RV_REJECT_ANY</td>
<td>Reject polygon if any one of the path codes are found</td>
</tr>
<tr>
<td>DVE_RV_REJECT_ALL</td>
<td>Reject polygon if exactly all the path codes are found</td>
</tr>
</tbody>
</table>

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” on page 5-11

**Example 1**

dcl lyrCond = dve_import_layer ("cond");
dcl lyrCond2 = dve_import_layer ("cond2");
dcl lyrError101 = dve_export_layer ("error101");
Example 2

Consider the recognition of a MOS gate:

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

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

```
clarlyPoly = dve_bool_or (lyrPoly, lyrPoly2);
lyrPolyOverlapping = dve_bool_and (lyrPoly, lyrPolyOverlap);
lyrPoly = dve_drc (poly_edge_code (lyrPolyOverlap),
                  DVE_RN_EDGE_SELECT, DVE_RV_ACCEPT_ANY,
                  DVE_RN_PATH_CODE, DVE_RV_INT);
lyrError101 += dve_drc (all_edges (lyrPoly), "Conductive metal overlaps");
```

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);
```
Polygon Selection

```c
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);
lyrError101 += dve_drc(all_edges(lyrGoodGate ), "Good Gate");
lyrError102 += dve_drc(all_edges(lyrBadGate ), "Bad Gate");
```
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()” on page 4-2

Syntax

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

where:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inLayer</td>
<td>A polygon layer produced by a merge operation between two layers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>operator</td>
<td>&lt;  Less than</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;= Less than or equal to</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>== Equal to</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt; Greater than</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt;= Greater than or equal to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>A real value in layout units</td>
<td></td>
<td></td>
</tr>
<tr>
<td>qualifierName, qualifierValue</td>
<td>A name, value pair that qualifies the rule</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Path Count Qualifier: DVE_RN_PATH_COUNT

Qualifier Resource Value:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DVE_RV_PATH_COUNT</td>
<td>(default) Select based upon path count of top polygon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVE_RV_ANTI_PATH_COUNT</td>
<td>Select based upon path count of bottom polygon</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Path Code Qualifiers

“DVE_RN_PATH_CODE” on page 5-11

Example 1

decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrError101 = dve_export_layer ("error101");
decl lyrPolyMerge = NULL;
decl lyrPoly = NULL;
Polygon Selection

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);
lyrError101 += dve_drc (all_edges (lyrPoly),
    "Metal layer outside and butting internally");

**Example 2**

See poly_path_length(), “Example 2” on page 5-18
**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()” on page 4-2

**Syntax**

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

where:

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

**Path Type Qualifier: DVE_RN_PATH_LENGTH**

Qualifier Resource Value

- **DVE_RV_MIN_PATH** Select based upon minimum path length of top polygon
- **DVE_RV_MAX_PATH** Select based upon maximum path length of top polygon
- **DVE_RV_TOTAL_PATH** Select based upon total path length of top polygon
- **VE_RV_MIN_ANTI_PATH** Select based upon minimum path length of bottom polygon
- **DVE_RV_MAX_ANTI_PATH** Select based upon maximum path length of bottom polygon
- **DVE_RV_TOTAL_ANTI_PATH** Select based upon total path length of bottom polygon

**Path Code Qualifiers**

“DVE_RN_PATH_CODE” on page 5-11
Polygon Selection

Example 1

decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrError101 = dve_export_layer ("error101");
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);
lyrError101 += 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);

![Diagram of transistor channel](image.png)

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 greater 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);

lyrError101 += dve_drc(all_edges(lyrSides ), "long channel");
To select 'thin' channels:

```plaintext
lyrSides = dve_drc(poly_path_length(lyrChannel) < 2, DVE_RN_PATH_CODE, DVE_RV_TIB, DVE_RN_PATH_LENGTH, DVE_RV_MIN_ANTI_PATH);
```

```plaintext
lyrError101 += dve_drc(all_edges(lyrSides), "thin channel");
```

Badly formed channels can be rejected using `poly_path_count()`:

```plaintext
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()” on page 5-21.
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()” on page 4-2

Syntax

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

where:

- inLayer1, inLayer2: A polygon layers
- qualifierName, qualifierValue: A name, value pair that qualifies the selection

Selection Qualifier: DVE_RN_INTER_SELECT

Qualifier Resource Value

- DVE_RV_ACCEPT: Select polygon if any path codes are found
- DVE_RV_REJECT: Reject polygon if any one of the path codes are found

Poly Code Qualifiers: DVE_RN_INTER_CODE

Qualifier Resource Value

- DVE_RV_INSIDE_ONLY: (default) The contained top is completely inside bottom and does not touch the inside of bottom.
  Pass: TIB
  Fail: TOP, INT, EXT, ENC
- DVE_RV_INSIDE_TOUCH: The contained top does touch the inside of bottom.
  Pass: TIB, INT
  Fail: TOP, EXT, ENC
Polygon Selection

**DVE_RV_INSIDE**
- **DVE_RV_INSIDE_ONLY or DVE_RV_INSIDE_TOUCH**
  - Top is inside the bottom.
  - Pass: TOP, EXT
  - Fail: TIB, INT, ENC

**DVE_RV_OUTSIDE_ONLY**
- Top is completely outside the bottom and does not touch the outside of the bottom.
- Pass: TOP
- Fail: TIB, INT, EXT, ENC

**DVE_RV_OUTSIDE_TOUCH**
- Top does touch the outside of the bottom.
- Pass: TOP, EXT
- Fail: TIB, INT, ENC

**DVE_RV_OUTSIDE**
- DVE_RV_OUTSIDE_ONLY or DVE_RV_OUTSIDE_TOUCH

**DVE_RV_CUT_ONLY**
- Top is partly inside the bottom and partly outside the bottom with no internal butt with the bottom, that is, it does not touch the inside of the bottom.
- Pass: TIB, TOP
- Fail: INT, ENC

**DVE_RV_CUT_TOUCH**
- Top is partly inside the bottom and partly outside the bottom and does internal butt (touch) the bottom.
- Pass: TIB, TOP, INT
- Fail: ENC
Example

decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrError101 = dve_export_layer ("error101");

decl lyrPoly = NULL;

lyrPoly = dve_drc (poly_inter_layer (lyrCond, lyrCond2),
    DVE_RN_INTER_CODE, DVE_RV_OUTSIDE);

lyrError101 += 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);

---

Polygon Selection Based on Edge Relationships  5-23
lyrError101 += dve_drc (all_edges (lyrPoly),
    "Conductive metal outside and inside touch");
Chapter 6: Boolean Operations on Edges

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

- “all_edges()” on page 6-2
- “invert_edges()” on page 6-3
Boolean Operations on Edges

all_edges()

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

See also: “dve_drc()” on page 4-2

Syntax

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

where:

\[\text{inLayer}\quad \text{A polygon layer}\]

\[\text{msgString}\quad \text{A string value that will be attached to the selected error segments}\]

Example

dcl lyrCond2 = dve_import_layer("cond2");
dcl lyrError101 = dve_export_layer("error101");

dcl lyrWork = NULL;

lyrWork = dve_drc(poly_area(lyrCond2) < 10.0);

lyrError101 += dve_drc (all_edges (lyrWork), "Conductive metal area < 10.0")
invert_edges()

Deselects selected edges and simultaneously selects unselected edges.

See also: “dve_drc()” on page 4-2

Syntax

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

where:

\[\text{inLayer} \quad \text{A polygon layer}\]
\[\text{msgString} \quad \text{A string value that will be attached to the selected error segments}\]

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

\[\text{decl lyrCond = dve_import_layer ("cond");}\]
\[\text{decl lyrError101 = dve_export_layer ("error101");}\]

\[\text{decl lyrEdgesGap} = \text{dve_drc (single_clearance (lyrCond) \leq 12.0, DVE_RN_POLARITY, DVE_RV_OUTSIDE, DVE_RN_TEMPLATE, DVE_RV_OPPOSITE, DVE_RN_EDGEANGLES, DVE_RV_PARALLEL, DVE_RN_STRUCTURE, DVE_RV_DIFFPOLYGON);}\]

\[\text{Next, use invert_edges to invert the error segments: good become bad and bad become good.}\]
\[\text{decl lyrEdgesInvert} = \text{dve_drc (invert_edges (lyrEdgesGap));}\]
Boolean Operations on Edges

Select opposite edges inside the paths:

```plaintext
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:

```plaintext
decl lyrPoly = dve_quadout (lyrEdges); 
lyrError101 = dve_drc (all_edges (lyrPoly), "Parallel interconnect < 12.0");
```
Chapter 7: Operations for Polygon Extraction from Edges

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

- “dve_plgout()” on page 7-2
- “dve_quadout()” on page 7-3
Operations for Polygon Extraction from Edges

*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()" on page 7-3

**Syntax**

dve_plgout (edgeLayer);

where:

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

**Example**

decl lyrCond = dve_import_layer("cond");
decl lyrError101 = dve_export_layer("error101");

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);

lyrError101 += 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()" on page 7-2

**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");
dcl lyrError101 = dve_export_layer ("error101");

decl lyrEdges = NULL;
dcl lyrPoly = NULL;
dcl 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);

lyrError101 += 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:

```plaintext
decl lyrPoly = dve_import_layer("poly");
decl lyrDiff = dve_import_layer("diff");
decl lyrError101 = dve_export_layer("error101");
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;
```
Start by undersizing the original polygon...

\[ \text{lyrShrunkPoly} = \text{dve_undersize}(\text{lyrPoly}, 0.5); \]

... to produce a new gate polygon:

\[ \text{lyrNewGate} = \text{dve_bool_and}(\text{lyrShrunkPoly}, \text{lyrDiff}); \]
\[ \text{lyrError101} += \text{dve_drc}(\text{all_edges}(\text{lyrNewGate}), "\text{new gate}"); \]

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

\[ \text{lyrBigDiff} = \text{dve_oversize}(\text{lyrDiff}, 1.0); \]

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

\[ \text{lyrPolyExtension} = \text{dve_bool_not}(\text{lyrPoly}, \text{lyrBigDiff}); \]

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

\[ \text{lyrBigPolyExtension} = \text{dve_drc}(\text{poly_area}(\text{lyrPolyExtension}) > 3.0); \]
\[ \text{lyrError102} += \text{dve_drc}(\text{all_edges}(\text{lyrBigPolyExtension}), "\text{lyrBigPolyExtension}"); \]
Operations for Polygon Extraction from Edges

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. Rebuild the 2 micron extension at the new channel length.

\[ \text{lyrOldGate} = \text{dve\_bool\_and(lyrPoly, lyrDiff)}; \]
\[ \text{lyrOversizedOldGate} = \text{dve\_oversize(lyrOldGate, 2.0)}; \]

Then use a nest clearance rule to extract the result:

\[ \text{lyrNestEdges} = \text{dve\_drc(nests(lyrNewGate, lyrOversizedOldGate) <= 2.0, DVE_RN\_TEMPLATE, DVE_RV\_OPPOSITE)}; \]
\[ \text{lyrPolyFiller} = \text{dve\_quadout(lyrNestEdges)}; \]
\[ \text{lyrError103} += \text{dve\_drc(all\_edges(lyrPolyFiller), "lyrPolyFiller")}; \]
Finally, just assemble all the bits and pieces of compensated poly:

```c
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:

```c
decl lyrDiel = dve_import_layer(\"diel\"));
```
Operations for Polygon Extraction from Edges

```c
decl lyrError101 = dve_export_layer("error101");

decl lyrTmp1, lyrTmp2, lyrTmp3 = NULL;

lyrTmp1 = dve_drc (width (lyrDiel) < 200.0);
lyrTmp2 = dve_drc (compensate (lyrTmp1, 5.0));
lyrTmp3 = dve_quadout (lyrTmp2);
lyrError101 = 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:

```c
lyrTmp1 = dve_drc(width(lyrDiel) < 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:

```c
lyrTmp1 = dve_drc(all_edges(diel));
lyrTmp2 = dve_drc(compensate (lyrTmp1, 1.0));
lyrTmp3 = dve_quadout (lyrTmp2);
lyrError101 = 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.
Chapter 8: Merge Operations on Polygons

This chapter describes boolean operations on polygons:

- “dve_bool_and()” on page 8-2
- “dve_bool_not()” on page 8-3
- “dve_bool_or()” on page 8-4

and the commands for combining layers.

- “dve_merge()” on page 8-5
- “dve_self()” on page 8-6
- “dve_self_merge()” on page 8-7
Merge Operations on Polygons

**dve_bool_and()**

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

**Syntax**

dve_bool_and (inLayer1, inLayer2);  

where:

- **inLayer1**: A polygon layer
- **inLayer2**: A polygon layer

**Example**

decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrError101 = dve_export_layer ("error101");

decl lyrPoly = NULL;

lyrPoly = dve_bool_and (lyrCond, lyrCond2);

lyrError101 += 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:

- **inLayer1** A polygon layer
- **InLayer2**

**Example**

```d
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrError101 = dve_export_layer ("error101");

decl lyrPoly = NULL;
lyrPoly = dve_bool_not (lyrCond, lyrCond2);
lyrError101 += dve_drc (all_edges (lyrPoly),
                      "Conductive metal not overlapping");
lyrPoly = dve_bool_not (lyrCond2, lyrCond);
lyrError101 += dve_drc (all_edges (lyrPoly),
                      "Conductive metal not overlapping");
```

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

**dve_bool_or()**
Merges overlapping shapes on a given layer. Returns: A polygon layer.

**Syntax**
outLayer = dve_bool_or (inLayer1 [, inLayer2]);

where:

\[
\begin{align*}
\text{inLayer1} & \quad \text{A polygon layer} \\
\text{inLayer2} & \quad \text{A polygon layer}
\end{align*}
\]

**Example**

```c
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrError101 = dve_export_layer ("error101");

decl lyrPoly = NULL;

lyrPoly = dve_bool_or (lyrCond, lyrCond2);

lyrError101 += 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:

  * inLayer1,   A polygon layer that is not the result of a dve_merge
  * inLayer2,
  * inLayerN

Example

decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrDiel = dve_import_layer ("diel");
decl lyrError101 = dve_export_layer ("error101");

decl lyrMerge = NULL;

lyrMerge = dve_merge (lyrCond, lyrCond2, lyrDiel);

lyrError101 += dve_drc (corner_edges (lyrMerge, 1.5, 90.5, 360.0),
                      "Concave corner edges");
Merge Operations on Polygons

**dve_self()**

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

**Syntax**

dve_self(inLayer)

where:

\[
\begin{align*}
\text{inLayer1,} & \quad \text{A polygon layer} \\
\text{InLayer2} &
\end{align*}
\]

**Example**

decl lyrCond = dve_import_layer("cond");
decl lyrError101 = dve_export_layer("error101");
decl lyrCondMerged = NULL;

dec1 lyrCondMerged = dve_self(lyrCond);
lyrError101 += 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 ("Convention for TOP and BOTTOM layers" on page 5-9). This command is particularly useful when used with polygon selection commands ("Polygon Selection Based on Merge Properties" on page 5-9). Returns : A polygon layer.

Syntax

dve_self_merge (inLayer1, inLayer2)

where:

\[ \text{inLayer1, inLayer2} \]

\[ \text{A polygon layer} \]

Example

dcl lyrCond = dve_import_layer ("cond");
dcl lyrCond2 = dve_import_layer ("cond2");
dcl lyrError101 = dve_export_layer ("error101");

dcl lyrPolyMerge = NULL;
dcl 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);

lyrError101 += 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);

lyrError101 += dve_drc (all_edges (lyrPoly1),
        "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 polygon 1 butts against polygon 2 on the same layer, and polygon 2 butts against the outside of polygon 5 on a different layer.
Merge Operations on Polygons
Chapter 9: Sizing Operations on Polygons

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

- “dve_oversize()" on page 9-2
- “dve_undersize ()" on page 9-3
Sizing Operations on Polygons

**dve_oversize()**

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

**Syntax**

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

where:

- `inLayer` A polygon layer
- `size` A real value in Layout units of the amount by which the size is to be increased

**Example**

```plaintext
decl lyrCond = dve_import_layer ("cond");
decl lyrCond2 = dve_import_layer ("cond2");
decl lyrError101 = dve_export_layer ("error101");

decl lyrPoly = NULL;
decl lyrOversize = NULL;

lyrPoly = dve_drc (poly_inter_layer (lyrCond, lyrCond2),
DVE_RN_INTER_CODE, DVE_RV_OUTSIDE);
lyrError101 += dve_drc (all_edges (lyrPoly), "Conductive metal outside");

lyrOversize = dve_oversize(lyrPoly, 5);
lyrError101 += 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**

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

where:

- `inLayer`: A polygon layer
- `size`: A real value in Layout units of the amount by which the size is to be decreased

**Example**

```c
 decl lyrCond = dve_import_layer ("cond");
dcl lyrCond2 = dve_import_layer ("cond2");
dcl lyrError101 = dve_export_layer ("error101");

dcl lyrPoly = NULL;
dcl lyrUndersize = NULL;

lyrPoly = dve_drc (poly_inter_layer (lyrCond, lyrCond2),
DVE_RN_INTER_CODE, DVE_RV_OUTSIDE);
lyrError101 += dve_drc (all_edges (lyrPoly), "Conductive metal outside");

lyrUndersize = dve_undersize(lyrPoly, 5);
lyrError101 += dve_drc (all_edges (lyrUndersize), "Undersize Conductive metal outside");
```
Sizing Operations on Polygons
Chapter 10: Macros

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

- “intrusion()” on page 10-2
- “protrusion()” on page 10-4.
Macros

intrusion()

Checks the intrusion of BOTTOM layer into TOP layer (“Conventions for TOP and BOTTOM layers” on page 5-9).

Syntax

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

where:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorLayer</td>
<td>Layer with error segments</td>
</tr>
<tr>
<td>inLayer1</td>
<td>Polygon TOP layer</td>
</tr>
<tr>
<td>inLayer2</td>
<td>Polygon BOTTOM layer</td>
</tr>
<tr>
<td>operator</td>
<td>Less than, Less than or equal to, Equal to, Greater than, Greater than or equal to</td>
</tr>
<tr>
<td>distance</td>
<td>A distance value in layout units</td>
</tr>
<tr>
<td>msgString</td>
<td>A string value that will be attached to the error segments</td>
</tr>
</tbody>
</table>

Method

dcl mergeGate = dve_bool_and (inLayer2, inLayer1);

dcl selectGate = dve_drc (poly_edge_code (mergeGate), DVE_RN_EDGE_SELECT, DVE_RV_ACCEPT_ANY, DVE_RN_PATH_CODE, DVE_RV_BIT);

dcl overlapGate = dve_bool_and (selectGate, inLayer2);
dcl 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

dcl lyrCond = dve_import_layer ("cond");
dcl lyrCond2 = dve_import_layer ("cond2");
dcl lyrError101 = dve_export_layer ("error101");

lyrError101 = dve_drc (intrusion (lyrCond2, lyrCond) <= 2.0);
cond

intrusion < 2.0

cond2
Macros

protrusion()
Checks the extension of TOP layer out of BOTTOM layer (“Convention for TOP and BOTTOM layers” on page 5-9).

Syntax

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

where:

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

Method

dcl gatePoly = dve_drc (poly_inter_layer (inLayer1, inLayer2), DVE_RN_INTER_CODE, DVE_RV_CUT);

dcl 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

dcl lyrCond = dve_import_layer("cond");
dcl lyrCond2 = dve_import_layer("cond2");

dcl lyrError101 = dve_export_layer("error101");

lyrError101 = dve_drc (protrusion (lyrCond, lyrCond2) <= 2);
protrusion < 2.0
cond2
Macros
Chapter 11: Troubleshooting

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.
Troubleshooting

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.
Troubleshooting

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.
Troubleshooting
Index

A
all_edges() command, 6-2

C
commands
  conditional selection
    DRC command description, 4-1
    edge based on clearance, 4-5
    edge based on corners, 4-36
    edge based on grid, 4-40
    polygon by edge relationships, 5-20
    polygon by intrinsic properties, 5-2
    polygon by merge properties, 5-9
  layer management, 3-1
  operations on edges
    edge compensation, 4-42
    polygon extraction from edges, 7-1
  operations on polygons
    merge, 8-1, 9-1
    usage errors, 11-2
    usage warnings, 11-4
  See Also operations
  compensate operation, 4-43
  compile
  errors and warnings, 11-1
  conditional selections
    DRC command description
      dve_drc(), 4-2
    edge based on clearance
      contains, 4-14
      double_clearance, 4-16
      external, 4-18
      gap, 4-6
      internal, 4-20
      nests, 4-22
      notch, 4-7
      single_clearance, 4-8
      spacing, 4-10
      width, 4-12
    edge based on corners
      corner_edges, 4-37
    edge based on grid
      offgrid(), 4-41
    edge by select all or inversion
      all_edges(), 6-2
      invert_edges(), 6-3
      polygon based on edge relationships
        poly_inter_layer(), 5-21
      polygon based on intrinsic properties
        poly_area, 5-3
        poly_hole_count, 5-4
        poly_line_length, 5-7
        poly_perimeter, 5-8
      polygon based on merge properties
        overview, 5-9
        poly_edge_code(), 5-12
        poly_path_count(), 5-15
        poly_path_length(), 5-17
    contains command
      description, 4-14
      example, 2-11
    corner_edges command, 4-37
    custom DRC example, 1-5

D
design layers, an example, 2-7
design rules
  directories, 1-2
  example, 2-2
  writing, 2-1
directories
  for saving design rules, 1-2
  system examples, 1-14
double_clearance command, 4-16
dve_bool_and() operation, 8-2
dve_bool_not() operation, 8-3
dve_bool_or operation, 8-4
dve_combine() operation, 4-4
dve_drc() command, 4-2
dve_export_layer() command, 3-3
dve_import_layer(), 3-2
dve_merge() operation, 8-5
dve_oversize(), 9-2
dve_plgout() operation, 7-2
dve_quadout() operation, 7-3
DVE_RN_ANGLE_TOLERANCE, 4-24
DVE_RN_EDGE_ANGLES, 4-24
DVE_RN_POLARITY, 4-24
DVE_RN_POLARITY_FROM, 4-24
DVE_RN_POLARITY_TO, 4-24
DVE_RN_SEPARATE, 4-25
DVE_RN_SLOPE, 4-26
DVE_RN_SLOPE_FROM, 4-26
DVE_RN_SLOPE_TO, 4-26
DVE_RN_STRUCTURE, 4-25
DVE_RN_TOUCH, 4-28
dve_segsizel(), 4-48
dve_self(), 8-6
dve_self_merge() operation, 8-7
dve_undersize (), 9-3

e
edge commands
  based on clearance, 4-5
  based on corners, 4-36
  based on grid, 4-40
  relationships, polygons, 5-20
edge operations
  edge compensation
    compensate, 4-43
  polygon extraction from edges
dve_plgout(), 7-2
  dve_quadout(), 7-3
Edge Qualifiers, 4-24
error layers example, 2-7
errors
  command usage, 11-2
  layer management, 11-1
examples
  contains command, 2-11
  copying, 1-14
  custom DRC, 1-5
  design layers, 2-7
  design rule, 2-2
  directory, 1-14
  error layers, 2-7
  external command, 2-10
  in the program, 1-14
  internal command, 2-13
  nests command, 2-12
  performing quick DRC, 1-3
  poly_path_length command, 2-14
  polygon selection, 2-14
  rule registry file, 1-2
  spacing command, 2-9
  width command, 2-8
exclusion, definition of, 2-1
export layers description, 2-4
extension
  defined, for design rules, 2-1
external command
  description, 4-18
  example, 2-10
F
files
  example of simple design rule, 2-2
  rule registry example, 1-2
G
gap command, 4-6
I
import layers, description, 2-3
internal command
  description, 4-20
  example, 2-13
intrusion, definition of, 2-1
invert_edges() command, 6-3
L
layer management
  commands
    dve_export_layer(), 3-3
  errors, 11-1
  warnings, 11-2
layers
  design, and example, 2-7
  error, an example, 2-7
  export, description of, 2-4
  import, description of, 2-3
  management of, 2-3
  work, description of, 2-5
M
merging
  operations on polygons, 8-1, 9-1
  properties as basis of polygon selection, 5-9
N
nests command
  an example, 2-12
  description, 4-12
  notch command, 4-7
O
offgrid() command, 4-41
operations
  on edges, 7-1
  on polygons, 8-1, 9-1
See Also commands

P
Performing Boolean Operations, 8-9
poly_area command, 5-3
poly_edge_code() command, 5-12
poly_hole_count command, 5-4
poly_inter_layer() command
description, 5-21
examples
  poly_inter_layer command, 2-16
poly_line_length command, 5-7
poly_path_count() command, 5-15
poly_path_length command
description, 5-17
example, 2-14
poly_perimeter command, 5-8
polygon commands
  based on edge, 5-20
  based on intrinsic properties, 5-2
  based on merge properties, 5-9
polygon operations
  merge
    dve_bool_and(), 8-2
    dve_bool_not(), 8-3
    dve_bool_or(), 8-4
    dve_combine(), 4-4
    dve_merge(), 8-5
    dve_self_merge(), 8-7
polygons
  selection example, 2-14
  projects
    copying, 1-14

Q
quick DRC
  example of performing, 1-3
  saving rule, 1-11

R
rule registry file example, 1-2
rules
clearance, an example, 2-8
conjunction, 2-18
directories, 1-2
saving quick rule, 1-11

S
selecting
  conditional, 4-1
single_clearance command, 4-8
spacing command
description, 4-10
example, 2-9

template Control, 4-30
TOP and BOTTOM layers, 5-9

U
Using edge codes, 5-9

W
warnings
  command usage, 11-4
  layer management, 11-2
width command
description, 4-12
example, 2-8
work layers, description, 2-5