

SIEMENS EDA

Calibre® Mask Data Preparation User's and Reference Manual

Software Version 2021.2

SIEMENS

Unpublished work. © 2021 Siemens

This material contains trade secrets or otherwise confidential information owned by Siemens Industry Software, Inc., its subsidiaries or its affiliates (collectively, "Siemens"), or its licensors. Access to and use of this information is strictly limited as set forth in Customer's applicable agreement with Siemens. This material may not be copied, distributed, or otherwise disclosed outside of Customer's facilities without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This document is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made. Siemens disclaims all warranties with respect to this document including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement of intellectual property.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. Siemens' **End User License Agreement** may be viewed at: www.plm.automation.siemens.com/global/en/legal/online-terms/index.html.

No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

TRADEMARKS: The trademarks, logos, and service marks ("Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Support Center: support.sw.siemens.com

Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Table of Contents

Chapter 1	
Introduction to Mask Data Preparation.....	11
Calibre MDP Overview	11
Calibre MDP Workflow.....	14
Calibre MDP Prerequisites	17
Syntax Conventions	17
Chapter 2	
Calibre FRACTURE Operations	19
Calibre FRACTURE Workflow	19
Calibre FRACTURE Invocation	21
Calibre FRACTURE Syntax	23
Global Calibre FRACTURE Syntax.....	24
Global Calibre FRACTURE Arguments	29
Common Arguments for Vector Beam Formats.....	40
FRACTURE HITACHI (FRACTUREh)	44
FRACTURE JEOL (FRACTUREj)	49
FRACTURE MEBES (FRACTUREm)	55
FRACTURE MICRONIC (FRACTUREc)	60
FRACTURE NUFLARE (FRACTUREt)	66
VSB11 Format	67
VSB12 V1 Format	73
VSB12i Format.....	80
VSB12 V2 Format	88
FRACTURE NUFLARE_MBF (FRACTUREn).....	95
FRACTURE OASIS_MAPPER (FRACTUREp)	101
FRACTURE OASIS_MASK (FRACTUREv).....	105
FRACTURE OASIS_MBW (FRACTUREi)	110
Considerations for Calibre FullScale Runs	115
Chapter 3	
Section-Based Processing, Calibre Embedded SVRF, and MDP EMBED	117
Section-Based Processing	117
Embed SVRF Commands in FRACTURE or MDPVERIFY Statements.....	119
Embed SVRF Commands in an MDP EMBED Statement	123
MDP EMBED.....	126
SVRF Command Support in Embedded SVRF.....	138
Influence the Interaction Range (BOUND Keyword).....	143
Converting a Rule File to Embedded SVRF	145
Embedded SVRF With Multiple Outputs in Calibre MDPverify and MDP EMBED	147
MDP EMBED Output to HDB and DDO	148
Processor Counts for MDP EMBED in the Transcript	150

Chapter 4	
Preprocessing With SVRF	153
Environment Setup	154
Layout Statements	154
Unit Statements	155
Layer Specification Statements	156
Results Reporting	156
Error Handling	157
Data Manipulation	159
Layer Selectors	159
Layer Constructors	160
Layer Creation Statements	161
Key Concepts Within SVRF	163
Calibre Results	164
The Results Database	164
The DRC Results Summary	164
The Calibre Transcript	164
Calibre Layer Data	165
Edge and Polygon Data	166
Layer of Origin	167
Understanding Dimension Checks	170
Identify Edge Pairs	171
Create Measurement Regions	173
Return Intersection Edges	175
Understanding the Size Operation	176
Examples of Common Pre-Processing	179
Border-Aware Sizing	179
Tone-Reversal	180
Chapter 5	
FRACTURE Examples	181
MEBES Example Using Magnify and Rotate	181
JEOL Example	181
VSB11 Example	182
HITACHI Example	184
Micronic Example	185
OASIS.MASK Example	186
Chapter 6	
Calibre MDPverify	189
Calibre MDPverify Inputs	189
Calibre MDPverify Outputs	190
MDPVERIFY	191
Region and Shift Default Computation	205
Data Transformation	206
Compare Fractured Data to the Layout Database	207
Compare Fractured Data to Fractured Data	208
Comparing Two VSB Job Decks	215

Table of Contents

Using Proper Shift Values	218
Automatic Shift Alignment	218
Manual Shift Alignment.....	219
Avoiding Problems With Shifting	220
Define the Verification Region.....	221
Examples	223
Example 1 — MEBES to an Internal Calibre Layer	223
Example 2 — MEBES to MEBES	224
Example 3 — MEBES to JEOL	225
Chapter 7	
Calibre MDPmerge.....	227
Multithreaded Operations	227
Calibre MDPmerge Limitations.....	228
Supported Job Deck Values.....	228
MDPMERGE.....	229
View the Calibre MDPmerge Results	232
Chapter 8	
Calibre MDPstat.....	235
Calibre MDPstat Reporting	235
MDPSTAT.....	237
Chapter 9	
Calibre Rule-Based MPC and MDP Utility Commands.....	245
Calibre MPCpro Workflow	245
DENSITY CONVOLVE SVRF Statement	248
DENSITY CONVOLVE	250
MDP MAPSIZE SVRF Statement.....	267
MDP MAPSIZE	270
MDP CHECKMAP SVRF Statement	272
MDP CHECKMAP.....	273
MDP OASIS_EXTENT SVRF Statement.....	276
MDP OASIS_EXTENT	277
MDP DATAPREP SVRF Statement.....	281
MDP DATAPREP	282
Chapter 10	
Calibre MASKOPT	287
The Calibre MASKOPT Setup File.....	287
Calibre MASKOPT Commands	290
LITHO MASKOPT	291
setlayer mdp_maskopt	294
direct_input.....	296
direct_output.....	298

Appendix A	
Odd-Multiple Grid Sizing	301
Understanding Odd-Multiple Grid Sizing	301
Adjustments for Odd-Multiple Grid Sizing	302
Appendix B	
List of Exceptions	307
Transcript Messages	307
Status Values	308
Appendix C	
MDP Best Practices	311
General Recommendations for MDP Tools	312
Input Preparation for DDE	312
Indexing and Injection for OASIS Files	313
Parallel DDE Run Recommendations	313
Reduce Data Size to Address RAM Issues	314
Global Fracture Best Practices	315
Common Fracture Settings	315
Avoid Bottlenecks During Fracture	316
VSB Fracture and Calibre MDPview	317
Calibre FRACTUREt Best Practices	317
MDP EMBED Best Practices	317
DENSITY CONVOLVE Best Practices	318

Index**Third-Party Information**

List of Figures

Figure 1-1. Complete Process Flow on the Calibre Platform	15
Figure 2-1. INSIDE OF Coordinates	25
Figure 2-2. Regions Define the Portion of the Layer to Fracture	26
Figure 2-3. Internal Critically Dimensioned Portion of a Design	41
Figure 2-4. External Critically Dimensioned Feature	41
Figure 2-5. Critical Features in a Normal Tone Fracture	42
Figure 2-6. Stairstep Approximation (Hitachi)	46
Figure 2-7. Micronic Origin Shift	64
Figure 2-8. Micronic Origin Shift Continued	65
Figure 2-9. Stairstep Approximation (VSB11)	69
Figure 2-10. Stairstep Approximation (VSB12)	76
Figure 2-11. Stairstep Approximation (VSB12i)	82
Figure 2-12. Stairstep Approximation (VSB12_V2)	91
Figure 3-1. Hierarchical Versus Section-Based Processing	118
Figure 3-2. MDP EMBED — Sectioning the Data	124
Figure 3-3. MDP EMBED — Processing the SVRF	125
Figure 3-4. BOUND and Long Range SVRFs	145
Figure 4-1. How Polygon Data Divides Space	166
Figure 4-2. How Edge Data Divides Space	167
Figure 4-3. Reference Data Dependence on Layer of Origin	168
Figure 4-4. The Effect of Reference Data on Dimension Checks	169
Figure 4-5. Edge Pairs Measured by the Internal Operation	171
Figure 4-6. Edge Pairs Measured by the External Operation	172
Figure 4-7. Edge Pairings Measured by the Enclosure Operation	172
Figure 4-8. Measuring “Appropriate” Angles for Edge Pairings	173
Figure 4-9. Measurement Regions for Edge A	174
Figure 4-10. Creating Measurement Regions	176
Figure 4-11. Simple Oversizing	177
Figure 4-12. Sizing by an Odd Number of Grid Units	177
Figure 4-13. Truncating Spikes	178
Figure 4-14. Simple Shrink	179
Figure 4-15. Border-Aware Shrink	179
Figure 5-1. Fractured Data	184
Figure 6-1. SIZE Keyword Alternatives for Embedded SVRF	200
Figure 6-2. Comparing Fractured Data to the Layout Database	208
Figure 6-3. Fracture Transformations for Two Sets of Data	209
Figure 6-4. Calibre MDPverify Transformations for Two Sets of Data	210
Figure 6-5. Transforming Only One Set of Fractured Data	211
Figure 6-6. Comparing Two Databases With the Same Extent	213
Figure 6-7. Comparing Two Databases With Different Extents	215

Figure 6-8. INSIDE OF Regions Comparing VSB Job Decks	216
Figure 6-9. Default Shifting with One File	219
Figure 6-10. Calculating Shift Values	220
Figure 6-11. Verification Region Not Aligned With Original Data	221
Figure 6-12. Aligning Fractured Data With Original Data	221
Figure 7-1. Calibre MDPmerge Example	232
Figure 8-1. SMALLOUTSIDETRAP Examples	239
Figure 8-2. SPLITCD Examples	240
Figure 9-1. Examples of a Layer Before and After Density Convolution.....	248
Figure 9-2. TRUNCATE Keyword	252
Figure 9-3. WRAP Keyword	253
Figure 9-4. Data With Map Layers	267
Figure 9-5. MDP Mapsize SVRF Example	268
Figure 9-6. Soft Boundary Behavior	269
Figure 10-1. Illustration of MDP MASKOPT Arguments	295
Figure A-1. Off-Grid Data Caused by Odd-Multiple Sizing	301
Figure A-2. Snapping Corrects Off-Grid Data	302
Figure A-3. Center-Shifting After Snapping	302
Figure A-4. Shifting the Center by Adjusting the Region	303
Figure A-5. Border-Aware Sizing	304
Figure A-6. Border-Aware Odd-Multiple Shrinking	305

List of Tables

Table 1-1. Fracture Formats and Support	11
Table 1-2. Related Products and Their Manuals	15
Table 1-3. Syntax Conventions	17
Table 2-1. Hitachi Format Type Maximum File Size	46
Table 2-2. Output File Name Restrictions for Hitachi	48
Table 2-3. Grid Size Settings	51
Table 3-1. Brackets in Embedded SVRF	135
Table 3-2. Supported SVRF Statements for Embedded SVRF	139
Table 4-1. Layout Statements Summary	154
Table 4-2. Unit Statement Summary	155
Table 4-3. The Layer Specification Statements	156
Table 4-4. Calibre nmDRC Statements	157
Table 4-5. Flag Statements	158
Table 4-6. Commonly Used Layer Selector Operations	159
Table 4-7. Commonly Used Layer Constructor Operations	160
Table 4-8. The Layer Definition Statements	162
Table 4-9. Layer of Origin Example	167
Table 6-1. Inputs to Calibre MDPverify	190
Table 6-2. Effect of Transformation in FRACTURE and MDPVERIFY	206
Table 9-1. Mask Type and Inputs	284
Table 10-1. Calibre MASKOPT Commands	290
Table B-1. Calibre MDP Status Values	308

Chapter 1

Introduction to Mask Data Preparation

Calibre® Mask Data Preparation (MDP) is a series of command-line-based tools used to generate, correct, verify, and analyze photomask layouts.

Calibre MDP Overview	11
Calibre MDP Workflow	14
Calibre MDP Prerequisites	17
Syntax Conventions	17

Calibre MDP Overview

Calibre MDP tools are used to create data for mask writers, inspection, and metrology tools, and to evaluate the compliance and quality of a data set to the mask making limits prior to manufacturing.

Calibre MDP tools include:

- Calibre FRACTURE — Converts layer data into MEBES (Calibre® FRACTUREm™), JEOL (Calibre® FRACTUREj™), Hitachi (Calibre® FRACTUREh™), Micronic (Calibre® FRACTUREc™), Toshiba Nuflare (Calibre® FRACTUREt™ and Calibre® FRACTUREn™), OASIS.MASK (Calibre® FRACTUREv™), OASIS.MBW (Calibre® FRACTUREi™), and OASIS.MAPPER (Calibre® FRACTUREp™) formatted data. The supported formats are listed in [Table 1-1](#).

Table 1-1. Fracture Formats and Support

FRACTURE	Supported Formats
FRACTUREc	Micronic Version 1.6 and 1.8
FRACTUREj	JEOL Version 3.0: JBX-9000MV, JBX-9300FS-50kV, JBX-9300FS-100kV Version 3.0 and 3.1: JBX-3030MV, JBX-3040MV JBX-3200MV
FRACTUREm	MEBES Mode 3, 4 and 5 ¹

Table 1-1. Fracture Formats and Support (cont.)

FRACTURE	Supported Formats
FRACTUREh	Hitachi HL 800, HL900D, HL950M, HL 7000M
FRACTUREt	Nuflare, EBM 3000/3500/4000/6000/7000/8000/9000 VSB11, VSB12, VSB12i
FRACTUREn	NUFLARE_MBF
FRACTUREv	OASIS.MASK
FRACTUREi	OASIS.MBW
FRACTUREp	OASIS.MAPPER

1. For MEBES Mode 3, writing and indexing are not supported. Reading is supported for the following products and utilities: Calibre MDPview (including the utilities mebes2oasis, jobToOasis, and mebesinfo) and Calibre MDPverify.

- Calibre® MDP Embedded SVRF™ — Allows you to insert a block of Standard Verification Rule Format (SVRF) commands in a FRACTURE, Calibre MDPverify, MDP EMBED, or DENSITY CONVOLVE invocation. SVRF commands are documented in the [Standard Verification Rule Format \(SVRF\) Manual](#).
- Calibre® MDP EMBED™ — Provides section-based processing for some SVRF commands, essentially providing section-based DRC capability. Section-based processing is an alternate method of layout processing that may improve scalability and turnaround time for some limited cases such as mask rule checks (which are DRC-like checks on mask data).
- Calibre® MDPverify™ — Allows you to evaluate fractured MDP data by performing geometrical comparisons of fractured data output to the original layout data, database layer, or fractured data in a different pattern file.
- Calibre® MDPmerge™ — For VSB11 only, reads a VSB11 job deck and merges individual chips and chip placements into a single chip and placement, respectively, based on certain criteria. The output is a new job deck consisting of the merged chips and copies of chips that did not meet the merge criteria.
- Calibre® MDPstat™ — Analyzes vector beam data such as NUFLARE, JEOL, Hitachi, and OASIS®¹ formatted data allowing you to assess the quality of results of the fracture.
- Calibre® MPCpro™ — A rule-based MPC application that applies pattern-based mask process corrections specified by SVRF commands to your design data prior to

1. OASIS® is a registered trademark of Thomas Grebinski and licensed for use to SEMI®, San Jose. SEMI® is a registered trademark of Semiconductor Equipment and Materials International.

fracturing. This includes correcting linearity and proximity effects as well as longer-range processing effects such as fogging and develop- and etch-loading.

- Calibre® MASKOPT™— Modifies the input geometry to reduce shot count and improve mask write time through rule-based vertex alignment.
- Calibre® nmMPC™ and Calibre® nmCLMPC — Calibre nmMPC is a suite of functions used for modelling, simulating, and correcting distortions of the mask manufacturing process. Calibre nmCLMPC performs the same functions, but is specifically targeted at curvilinear layouts containing skew edge polygons. Models generated by this process are utilized by Calibre nmMPC, Calibre nmCLMPC, and FRACTURE tools to accurately reflect the final output of the mask process. Refer to the [Calibre nmMPC and Calibre nmCLMPC User's and Reference Manual](#) for further information.
- Calibre® MPCVerify — A grid-based mask process simulator and MPC results verification tool that is designed to predict mask manufacturing processes. It supports the industry-standard MPC usage models and seamlessly integrates with other Calibre tools. Refer to the [Calibre MPCVerify User's and Reference Manual](#) for further information.
- Calibre® MDPview™ — A Graphical User Interface (GUI) viewer that enables you to visually and interactively inspect the results of fracture and verify operations (pattern files, job decks), as well as large post-tape-out OASIS files. Calibre MDPview utilities are TCL-based commands that allow script-based manipulation of fractured data and job decks, and include conversion of pattern files or job decks to OASIS (job smashing), conversion of MEBES job decks from MEBES to other formats, and pattern file quality, optimization and informational utilities. This tool is documented in the [Calibre MDPview User's and Reference Manual](#).
- Calibre® Job Deck Editor — A tool invoked from Calibre MDPview that is used to generate an optimized chip placement on the wafer, with the ability to manually edit chips in the job deck. This tool is documented in the [Calibre Job Deck Editor User's Manual](#).
- Calibre® MDPDefectAvoidance™ — A tool invoked from Calibre MDPview that is used to find shifts in a layout to prevent extreme ultraviolet lithography (EUVL) blank mask defects from appearing on patterns. This tool is documented in the [Calibre MDPDefectAvoidance User's Manual](#).
- Calibre® DefectReview™ — A tool that provides efficient classification and trend analysis of defects identified by wafer and mask inspection systems. The software includes features that perform defect navigation, selection, filtering, classification, and clustering. Analysis tools include visual display capability, CD analysis, analysis over multiple inspections, and repeatability and trend analysis. This tool is documented in the [Calibre DefectReview User's Manual](#).

- Calibre® MDPAutoClassify™ — A tool that is used for automatic classification of defects observed on a blank substrate before any patterning is performed on the substrate. This tool is documented in the *Calibre MDPAutoClassify User's Manual*.
- Calibre® DefectClassify™ — A tool that enables automatic classification of defects observed on a patterned mask. This tool is documented in the *Calibre DefectClassify User's Manual*.

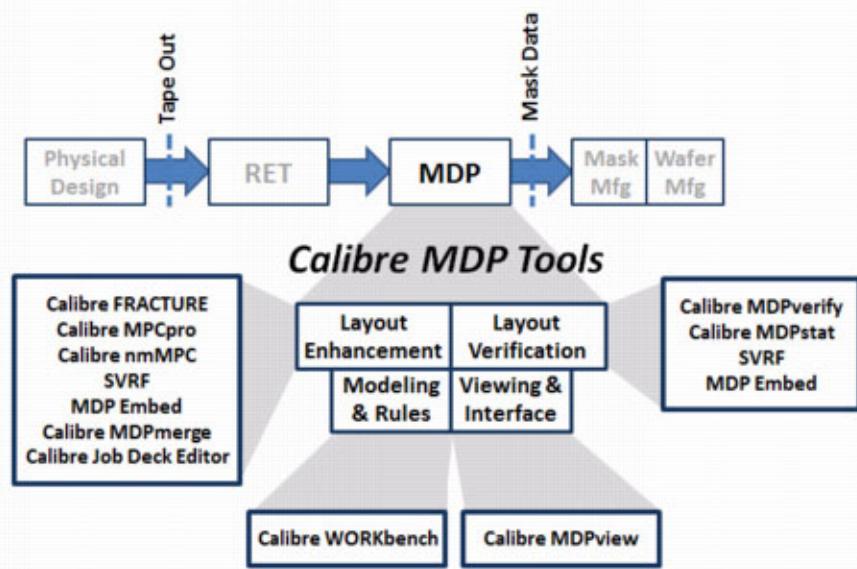
Each of the MDP utilities is invoked using a command-line syntax. The usage and requirements for each tool are documented in separate chapters of this manual.

- Chapter 2, “[Calibre FRACTURE Operations](#)” on page 19
- Chapter 3, “[Section-Based Processing, Calibre Embedded SVRF, and MDP EMBED](#)” on page 117
- Chapter 6, “[Calibre MDPverify](#)” on page 189
- Chapter 7, “[Calibre MDPmerge](#)” on page 227
- Chapter 8, “[Calibre MDPstat](#)” on page 235
- Chapter 9, “[Calibre Rule-Based MPC and MDP Utility Commands](#)” on page 245
- Chapter 10, “[Calibre MASKOPT](#)” on page 287

Calibre MDP Workflow

The Calibre MDP product line completes the integrated flow from IC design to IC mask manufacturing. The Calibre MDP tool suite allows for seamless continuation of the data manipulations required for Resolution Enhancement Techniques (RET) such as Phase Shift Mask (PSM), Scattering Bars (SB), and Optical Process Correction (OPC) to the mask data format conversion in one batch run.

Figure 1-1. Complete Process Flow on the Calibre Platform



Calibre's hierarchical and section-based geometry processing includes functions such as layer derivation, mirroring, scaling, rotation, planarization fill, global, and selective sizing. The flow concludes with the output of commonly-used mask writer formats for advanced mask-making in the subwavelength era, including Variable-Shaped Beam (VSB) formats, GDSII, and OASIS. Hierarchical and section-based mask rule checking (MRC) is supported as well as mask proximity correction (MPC).

Table 1-2 lists the documents associated with the related Calibre tools.

Table 1-2. Related Products and Their Manuals

Related Products	Documentation
All post-tapeout products	<i>Calibre Post-Tapeout Flow User's Manual</i>

Table 1-2. Related Products and Their Manuals (cont.)

Related Products	Documentation
Calibre® FRACTUREc™ Calibre® FRACTUREh™ Calibre® FRACTUREi™ Calibre® FRACTUREj™ Calibre® FRACTUREm™ Calibre® FRACTUREn™ Calibre® FRACTUREp™ Calibre® FRACTUREt™ Calibre® FRACTUREv™ Calibre® MDPmerge™ Calibre® MDPstat™ Calibre® MDPverify™ Calibre® MPCpro™ Calibre® MASKOPT™ Calibre® MDP Embedded SVRF	<i>Calibre Mask Data Preparation User's and Reference Manual</i> <i>Calibre Release Notes</i>
Calibre® nmMPC™ Calibre® nmCLMPC Calibre® nmOPC™	<i>Calibre nmMPC and Calibre nmCLMPC User's and Reference Manual</i> <i>Calibre nmOPC User's and Reference Manual</i>
Calibre® Interactive™ Calibre® RVE™	<i>Calibre Interactive User's Manual</i> <i>Calibre RVE User's Manual</i>
Calibre® MDPview™	<i>Calibre MDPview User's and Reference Manual</i> <i>Calibre DESIGNrev Layout Viewer User's Manual</i> <i>Calibre Release Notes</i>
Calibre® nmDRC™ Calibre® nmDRC-H™	<i>Calibre Release Notes</i> <i>Calibre Verification User's Manual</i> <i>Standard Verification Rule Format (SVRF) Manual</i>
Calibre® WORKbench™	<i>Calibre WORKbench User's and Reference Manual</i>
Tcl/Tk Batch Commands	<i>Calibre DESIGNrev Reference Manual</i>
Calibre® Metrology API (MAPI)	<i>Calibre Metrology API (MAPI) User's and Reference Manual</i>
Calibre® Metrology Interface (CMi)	<i>Calibre Metrology Interface (CMi) User's Manual</i>

Table 1-2. Related Products and Their Manuals (cont.)

Related Products	Documentation
Calibre® Job Deck Editor	<i>Calibre Job Deck Editor User's Manual</i>
Calibre® MDPDefectAvoidance™	<i>Calibre MDPDefectAvoidance User's Manual</i>
Calibre® MPCverify	<i>Calibre MPCverify User's and Reference Manual</i>
Calibre® DefectReview™	<i>Calibre DefectReview User's Manual</i>
Calibre® MDPAutoClassify™	<i>Calibre MDPAutoClassify User's Manual</i>
Calibre® DefectClassify™	<i>Calibre DefectClassify User's Manual</i>

Calibre MDP Prerequisites

There are a number of prerequisites before using Calibre MDP utilities.

- **Platform support** — Calibre MDP is available on all supported platforms found in the *Calibre Administrator's Guide*. Refer to that document for instructions on how to install Calibre software.
- **Licensing** — You must have valid licenses for your FRACTURE formats and MDP utilities, as well as a Calibre nmDRC license. For more information on licensing, refer to the *Calibre Administrator's Guide*.
- **Required files** — The following files are required to perform most Calibre MDP operations:
 - A GDS or OASIS layout ready for mask data preparation
 - An SVRF file

Syntax Conventions

The command descriptions use font properties and several metacharacters to document the command syntax.

Table 1-3. Syntax Conventions

Convention	Description
Bold	Bold fonts indicate a required item.
<i>Italic</i>	Italic fonts indicate a user-supplied argument.
Monospace	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.

Table 1-3. Syntax Conventions (cont.)

Convention	Description
<u>Underline</u>	Underlining indicates either the default argument or the default value of an argument.
UPPercase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[]	Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.
‘ ’	Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.
or	Vertical bars indicate a choice between items. Do not include the bars when entering the command.
...	Three dots (an ellipsis) follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.

Example:

```
DEvice {element_name [('model_name')]}

device_layer {pin_layer [('pin_name')] ...}

['<auxiliary_layer>' ...]

[('swap_list') ...]

[BY NET | BY SHAPE]
```

Chapter 2

Calibre FRACTURE Operations

The Calibre FRACTURE operation converts layer data into MEBES, JEOL, Hitachi, Micronic, Nuflare, NUFLARE_MBF, OASIS.MASK, OASIS.MAPPER, and OASIS.MBW formatted data. You define as input to the FRACTURE Standard Verification Rule Format (SVRF) a rectangular section of a Calibre polygon layer. The polygons on the layer are optionally modified and converted to trapezoids. The trapezoids are then transformed to new coordinate and unit systems, and rearranged into the hierarchy of the requested format. These trapezoids are then output in the fracture format.

The SVRF rule file you use to invoke the fracturing software must conform to all the requirements for any SVRF rule file. For a complete description of these requirements, refer to the Standard Verification Rule Format (SVRF) Manual.

The DRC RESULTS DATABASE statement must be included within the file and must specify a valid path, whether you output any layer data or not. Since the FRACTURE operation does not generate any layout database output, the output does not contain any polygon data unless you choose to output the original layer or data generated within the Calibre hierarchical engine.

Calibre FRACTURE Workflow	19
Calibre FRACTURE Invocation	21
Calibre FRACTURE Syntax.....	23
Global Calibre FRACTURE Syntax.....	24
Global Calibre FRACTURE Arguments	29
Common Arguments for Vector Beam Formats.....	40
FRACTURE HITACHI (FRACTUREh)	44
FRACTURE JEOL (FRACTUREj)	49
FRACTURE MEBES (FRACTUREm)	55
FRACTURE MICRONIC (FRACTUREc)	60
FRACTURE NUFLARE (FRACTUREt)	66
FRACTURE NUFLARE_MBF (FRACTUREn).....	95
FRACTURE OASIS_MAPPER (FRACTUREp)	101
FRACTURE OASIS_MASK (FRACTUREv).....	105
FRACTURE OASIS_MBW (FRACTUREi)	110
Considerations for Calibre FullScale Runs.....	115

Calibre FRACTURE Workflow

The basic workflow for using Calibre FRACTURE is comprised of two fundamental steps.

1. Write a FRACTURE command in an SVRF rule file to process the layers from your GDS or OASIS layout. The syntax is described in the section “[Calibre FRACTURE Syntax](#)” on page 23.
2. Execute the commands in the SVRF using the Calibre nmDRC hierarchical engine. Refer to “[Calibre FRACTURE Invocation](#)” on page 21 for more details.

Calibre FRACTURE software generates a single pattern file (such as JEOL, MEBES, OASIS.MASK, OASIS.MAPPER, OASIS.MBW, or Hitachi) or a directory (such as Nuflare) from a flat or hierarchical layer, either input or derived. The FRACTURE SVRF operation executes the software and generates pattern file output (for example, MEBES, JEOL, Hitachi, or Nuflare) from within the Calibre nmDRC hierarchical engine. It can operate with multithreading by including the -turbo and -turbo_litho switches in the Calibre invocation, and you can also utilize the distributed processing capability of Calibre® MTflex™ by including the -remotefile *filename* option. For more information on Calibre MTflex, refer to the [Calibre Administration Guide](#).

Calibre FRACTURE Invocation

The Calibre FRACTURE software is executed in the Calibre nmDRC hierarchical engine. You invoke this engine using the calibre command from a UNIX command line.

Usage

```
calibre {-drc -hier | -pto} [-remotefile filename] [-turbo [number_of_processors]] [-turbo_all]  
[-turbo_litho [number_of_processors]] rules
```

Arguments

- **-drc**
A required switch specifying DRC checking.
- **-hier**
A required switch invoking hierarchical DRC checking (Calibre nmDRC-H).
- **-pto**
Enables the Calibre® FullScale™ engine, a separately licensed Calibre engine that improves scalability and runtime for typical post-tapeout designs. Refer to “[Considerations for Calibre FullScale Runs](#)” on page 115 for information.
- **-remotefile *filename***
An optional argument that activates Calibre® MTFlex™ functionality, specifying information about the machines Calibre uses as remote and local hosts. This option activates distributed processing in Calibre. The argument for -remotefile is a file location that contains configuration information for the local and remote hosts, to which Calibre partitions the run, and information about launching the run.

The only difference from any other Calibre function is that fracturing uses temporary disk files. The directory in which these files are stored is specified by the “LOCAL HOST DIR...” line in the Calibre MTFlex configuration file.
- **-turbo [*number_of_processors*]**
Instructs Calibre nmDRC-H to use multithreaded parallel processing.

The optional *number_of_processors* argument is a positive integer that specifies the number of CPUs to use. If you do not specify a *number_of_processors*, Calibre nmDRC-H runs on the maximum number of CPUs available for which you have licenses.

Calibre nmDRC-H runs on the maximum number of CPUs available if you specify a number greater than the maximum available. For example:

```
calibre -drc -hier ... -turbo 3 ...
```

operates on two processors for a 2-CPU machine.

See “License Consumption for Distributed Calibre” in the [Calibre Administrator’s Guide](#) for additional information about license scaling with CPU count.

- **-turbo_all**

This option is used with the -turbo option. This option halts Calibre tool invocation if the tool cannot obtain the exact number of CPUs you specified.

For example:

```
calibre -drc -hier -turbo -turbo_all rule_file
```

executed on an 8-CPU machine for a hierarchical DRC MT run is the same as specifying this:

```
calibre -drc -hier -turbo 8 -turbo_all rule_file
```

Without -turbo_all, the Calibre tool normally uses fewer threads than requested if the requested number of licenses or CPUs is unavailable.

See “License Consumption for Distributed Calibre” in the *Calibre Administrator’s Guide* for additional information about license scaling with CPU count.

- **-turbo_litho [number_of_processors]**

Specifies the number of processors to use for RET and MDP applications. This is only specified with -turbo.

The optional *number_of_processors*. argument is a positive integer that specifies the number of CPUs to use for RET and MDP processes. If you do not specify *number_of_processors*, Calibre runs on the maximum number of CPUs available for which you have licenses, regardless of the -turbo_litho setting.

You must specify the -turbo and the -turbo_litho options concurrently in a single command line and the respective *number_of_processors* arguments can vary between the two options. Choosing the best values for *number_of_processors* for -turbo and -turbo_litho will depend on the contents and style of your rule file.

- **rules**

A required user-supplied argument specifying the path name for the SVRF rule file. This file may include DRC statements, fracture operations, and other Calibre operations.

For more information on multithreaded processing, refer to the *Calibre Verification User’s Manual* and the *Calibre Administrator’s Guide*.

Calibre FRACTURE Syntax

The Calibre FRACTURE operation converts layer data into MEBES, JEOL, Hitachi, Micronic, Nuflare, NUFLARE_MBF, OASIS.MASK, OASIS.MAPPER, and OASIS_MBW formatted data.

You define as input to the FRACTURE Standard Verification Rule Format (SVRF) a rectangular section of a Calibre polygon layer. The polygons on the layer are optionally modified and converted to trapezoids. The trapezoids are then transformed to new coordinate and unit systems, and rearranged into the hierarchy of the requested format. These trapezoids are finally output in the fracture format. See the additional arguments for the various supported formats.

Global Calibre FRACTURE Syntax	24
Global Calibre FRACTURE Arguments.....	29
Common Arguments for Vector Beam Formats.....	40
FRACTURE HITACHI (FRACTUREh)	44
FRACTURE JEOL (FRACTUREj).....	49
FRACTURE MEBES (FRACTUREm).....	55
FRACTURE MICRONIC (FRACTUREc).....	60
FRACTURE NUFLARE (FRACTUREt)	66
FRACTURE NUFLARE_MBF (FRACTUREn)	95
FRACTURE OASIS_MAPPER (FRACTUREp).....	101
FRACTURE OASIS_MASK (FRACTUREv)	105
FRACTURE OASIS_MBW (FRACTUREi).....	110

Global Calibre FRACTURE Syntax

All Calibre FRACTURE commands use a common syntax structure.

Usage

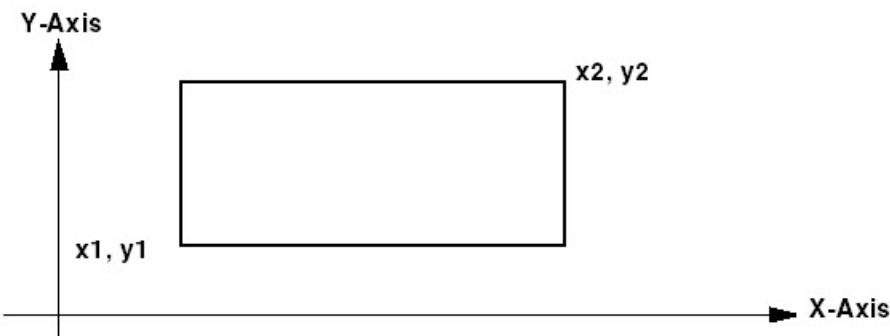
```
FRACTURE type layer [...layerN]  
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]  
[FILENAME output_file_name]  
{FILE {parameter_block_name | '['parameter_list '']}}
```

Arguments

- ***type***
Specifies a supported fracture format. Allowed arguments are HITACHI, JEOL, MICRONIC, NUFLARE, NUFLARE_MBF, OASIS_MASK, OASIS_MAPPER, and OASIS_MBW. Each of these formats has specific arguments that can optionally be specified in the *parameter_list*. Each of their descriptions are listed in these sections:
 - [FRACTURE HITACHI \(FRACTUREh\)](#)
 - [FRACTURE JEOL \(FRACTUREj\)](#)
 - [FRACTURE MEBES \(FRACTUREm\)](#)
 - [FRACTURE MICRONIC \(FRACTUREc\)](#)
 - [FRACTURE NUFLARE \(FRACTUREt\)](#)
 - [FRACTURE NUFLARE_MBF \(FRACTUREn\)](#)
 - [FRACTURE OASIS_MAPPER \(FRACTUREp\)](#)
 - [FRACTURE OASIS_MASK \(FRACTUREv\)](#)
 - [FRACTURE OASIS_MBW \(FRACTUREi\)](#)
- ***layer* [...*layerN*]**
A required string specifying the name of the input layer(s). This layer can be an original layer, read in using the SVRF LAYER statement, or it can be a derived layer created by manipulating one or more layers within the Calibre nmDRC hierarchical engine. You specify the portion of the input layer for output using the INSIDE OF keyword and its arguments using one of the following forms:
 - INSIDE OF EXTENT — Uses the extent of geometries on the first input *layer*. This is the default setting. However, this is not the recommended setting, since you need to carefully control the fracture region of the data with explicit coordinates in order to regulate the location of chip placement in the job deck.
 - INSIDE OF *x1 y1 x2 y2* — Defines a rectangular fracture region using x-y axis coordinates you supply. This is a recommended method for defining the fracture

region. The $x1$ and $y1$ coordinates specify the lower left corner $x2$ and $y2$ coordinates specify the upper right corner, as shown in [Figure 2-1](#):

Figure 2-1. INSIDE OF Coordinates



You must specify the coordinates relative to the origin of the Calibre database, and negative coordinates must be enclosed in parentheses. For example: “INSIDE OF (-123) 123 (-12) 456.”

- INSIDE OF LAYER *layer2* — Uses the extent of geometries on the specified layers. The value for *layer2* cannot be the same layer as *layer*. The specified layer can be an implicitly derived layer, for example:

```
FRACTURE JEOL lay1 INSIDE OF LAYER ( SIZE lay2 BY 0.05 ) ...
FRACTURE JEOL lay1 lay2 lay3 INSIDE OF LAYER ( COPY lay2 ) ...
```

This is a recommended method for defining the fracture region if *layer2* is specified using explicit coordinates of a dummy layer or polygon pair. For example:

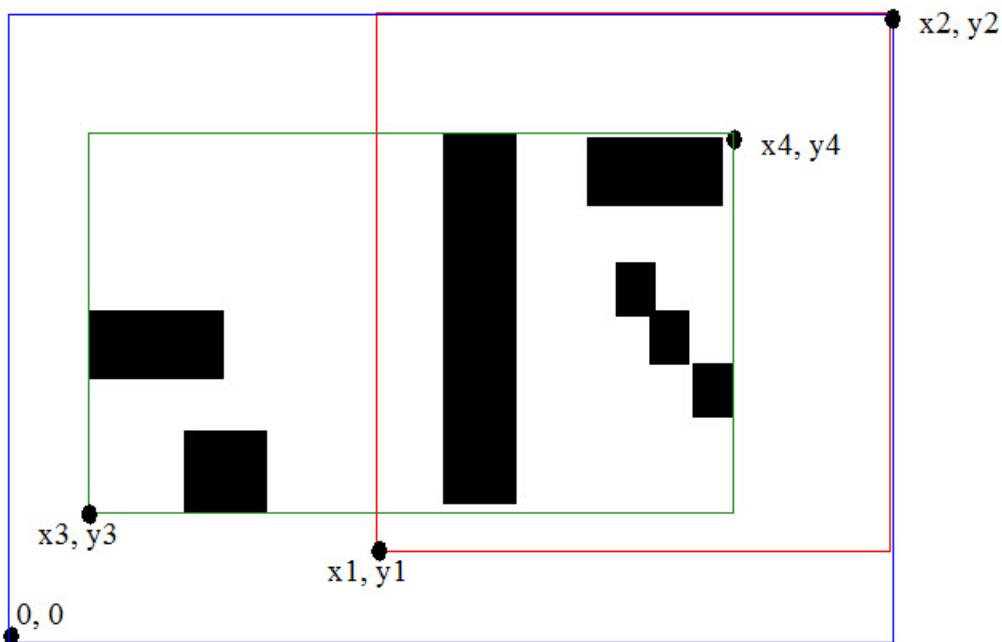
```
...
LAYER 999 fracture_extent
POLYGON 0 0 110000 130000 fracture_extent
FRACTURE JEOL lay1 INSIDE OF LAYER fracture_extent
```

Note

 If you omit the INSIDE OF keyword, the Calibre application uses the extent of *geometries* on the first input layer to the FRACTURE operation. This is equivalent to specifying INSIDE OF EXTENT.

[Figure 2-2](#) illustrates using fracture regions defined for the entire input layer explicitly defined as $(0, 0)$ - $(x2, y2)$, a subset of the input layer explicitly defined as $(x1, y1)$ - $(x2, y2)$, and the region of INSIDE OF EXTENT implicitly defined as $(x3, y3)$ - $(x4, y4)$.

Figure 2-2. Regions Define the Portion of the Layer to Fracture



Note

Layer data may be manipulated in the Calibre nmDRC hierarchical engine before you pass it to Calibre FRACTURE. If the input layer is a derived layer resulting from SVRF operations that included scaling (using the MAGNIFY operation) of an original layer, be sure to scale the region coordinates to compensate for the magnification applied to the hierarchical database. In such situations, you can adjust the hierarchical database unit size to match the final mask unit size. The unit size strongly influences coordinate snapping.

- **FILENAME** *output_file_name*

An optional keyword set specifying the name of the FRACTURE output file. The *output_file_name* argument is a user-supplied filename consisting of alphanumeric characters, underscores (_), and at most one period (.), which is part of the file extension (for example, .PF for a pattern file).

This keyword and the **file_name** (or **chip_directory**) argument are mutually exclusive (see the entry in the section “[Global Calibre FRACTURE Arguments](#)” on page 29); Calibre generates an error if you specify both for any one Calibre FRACTURE operation.

Note

The FILENAME keyword option is useful for scripting and automation; it accepts a variable name definition whereas **file_name** and **chip_directory** do not.

Use this keyword to specify the name of the fracture output file if you specify FRACTURE within a LITHO FILE statement.

The origin location of the fractured output file is based on the requirements of the Calibre FRACTURE format and the user-defined fracture region, regardless of the origin of the input data

- **FILE {parameter_block_name | '['parameter_list']}' }**

Specifies the method to control the type of information generated by a FRACTURE operation. See “Calibre FRACTURE Output With Fracture Arguments” in the Description section for further details.

Description

Calibre FRACTURE Input

The FRACTURE SVRF operation accepts one or more layers as inputs, specified by the *layer* option.

Calibre FRACTURE Output

The Calibre FRACTURE operation does not return any data to the Calibre hierarchical engine. However, the operation does output the fractured data, which it stores in either a single file or in a directory depending on the output format.

Unlike most other SVRF operations, the FRACTURE operation requires that users adhere to syntax restrictions. These restrictions apply only to the portion of the operation used to define the output, which follows the keyword FILE or is supplied using a reusable argument block

The output definition portion of the operation must be enclosed in square brackets ([]). The keywords and any associated values must be on lines strictly between the left and right brackets. That is, they cannot even be on the same line as either the left or the right bracket.

The output definition cannot contain the characters left bracket ([) or right bracket (]). The output definition can contain comments, which must begin with either the pound character (#) or a double forward slash (//). Either of these comment characters indicates that all text until the next new line is comment text.

Custom FRACTURE Output With Fracture Arguments

Format-specific arguments can be applied to control the type of information generated by a Calibre FRACTURE operation. The arguments are applied using the FILE keyword. You can supply the FRACTURE arguments that define the output in one of two ways:

- **Parameter Blocks**

You can define reusable parameter blocks using the SVRF LITHO FILE statement, then have them referenced by the Calibre FRACTURE operation (*parameter_block_name*). To create a parameter block, you supply the controls as a named block of text within the SVRF rule deck using the LITHO FILE statement. You reference a setup parameter block by specifying the block name as the argument to the FILE parameter in the fracture operation.

Note

 Use the **parameter_block_name** option if your rule file contains several FRACTURE operations that use the same parameters, allowing you to reuse code and ensure that all FRACTURE operations use the same parameters.

This example uses two parameter blocks named “fract1” and “fract2”:

```
LITHO FILE fract1 [
    <fracture parameter>
    <fracture parameter>
    ...
]
LITHO FILE fract2 [
    <fracture parameter>
    <fracture parameter>
    ...
]

VSB12_poly_data = {FRACTURE VSB12 poly INSIDE OF 0 0 10000 10000
    FILE fract1
}
VSB12_metal1_data = {FRACTURE VSB12 metal12 INSIDE OF 0 0 10000 10000
    FILE fract1
}
JEOL_poly_data = {FRACTURE JEOL poly INSIDE OF 0 0 10000 10000
    FILE fract2
}
```

Within the LITHO FILE statement, the controls must be enclosed within brackets ([]) and presented using the same conventions as when supplied inline. You can include comments within the section set off by square brackets, if you begin the comments with double forward slash (//) or a pound sign (#). These comment characters indicate that all text until the next newline is comment text.

- **Inline Definition of Parameters**

You can define FRACTURE parameters inline, as part of the FRACTURE operation in the SVRF file (**parameter_list**). You can define the parameters that define the fractured output as a list enclosed in brackets. Note the following syntax rules:

- You must specify the left bracket ([) before any of your definition parameters. You can specify only one parameter per line.
- You can not include any left- ([]) or right- ()) brackets within the parameters
- You can add commented text by using a double-forward slash (//), which specifies that all text up to the next newline character is comment text
- You must specify the right bracket (]) after the last of your definition parameters.

Global Calibre FRACTURE Arguments

There is a common list of arguments that apply to all FRACTURE formats.

Usage

```
FRACTURE type layer [...layerN]
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]
[FILENAME output_file_name]
--> {FILE {parameter_block_name} | '[' parameter_list '']}
```

The *parameter_list* options described in this section are global for all Calibre FRACTURE formats.

Arguments

- **file_name** *output_file_name*

A required argument specifying the name of the fracture output file.

output_file_name — a required user-supplied filename that consists of alphanumeric characters, underscores (_), and at most one period (.), which is part of the extension (for example, .PF for pattern files).

Tip

 You can use this argument to specify the name of the FRACTURE output file if you specify all the other arguments inline to the FRACTURE operation.

This argument and FILENAME are mutually exclusive; Calibre generates an error if you specify both for any one Calibre FRACTURE operation.

Note

 The **file_name** argument is not used for the **FRACTURE NUFLARE** (**FRACTUREt**) formats VSB11, VSB12, VSB12i and **FRACTURE NUFLARE_MBF** (**FRACTUREn**) because they output to a directory (**chip_directory**) instead.

- computation_mode {hier | section}

An optional argument set specifying the fracture method during processing.

hier — specifies the hierarchical processing method that improves performance when processing hierarchical input data. This is the default value.

section — specifies the section-based processing method, which improves scalability and performance when processing flat designs.

In some circumstances, there may be a slight variation between the outputs of hierarchical and section-based methods, in particular for skewed edges and off-grid vertices.

If you use Embedded SVRF (the <SVRFSTART> and <SVRFEND> block), then you must use section mode only.

- **input_region_severity {0 | 1}**

An optional argument set that controls the behavior of all fracturing formats when the input region is not a multiple of the input Database (DB) unit. If either one of the following conditions occur:

- Right region or left region x value is not a multiple of the input Database (DB) unit.
- Lower region or upper region y value is not a multiple of the input DB unit.

then the option performs actions based on the setting:

0 (default) — A warning message is issued and the program continues to run.

1 — An error message is issued and the program exits immediately.

This allows users to exercise more or less care with respect to fracture region precision and database precision consistency in order to be warned of potentially unexpected snapping.

- **index_options [-topcell *cellname*]**

An optional argument set that sets OASIS input module and indexing options. The -topcell option specifies the top cell name for indexing. This is identical to the oasisIndex [*topcellname*] utility in Calibre MDPview.

This argument set is used in cases where you have OASIS files with more than one top cell. The oasisIndex and the indexing capability of Calibre FRACTURE, Calibre MDPverify, MDP EMBED, and DENSITY CONVOLVE all assume that there is only one top cell. If there is more than one top cell, then oasisIndex and the other batch direct-data-entry tools by default pick the top cell that has the deepest hierarchy. If you want to pick a top cell that has a shallow hierarchy, you can instead specify the top cell name explicitly with -topcell.

- **lint**

An optional keyword that invokes a set of checks on the input and arguments of the FRACTURE command. This “lint screen” warns of potential performance problems. These problems are usually caused by poor hierarchical injection in the hierarchical database (HDB) or excessive geometry flattening in prior operations. The following is example output from the lint operation:

```
-----  
COMPUTATION MODE HIER selected by user.  
MDP LINT:  
MDP RUNTIME WARNING: MDP LINT; High top cell edge count (>10%);  
progress may be slow.  
MDP RUNTIME WARNING: MDP LINT; High edge count (>5%) in cell TOP;  
progress may be slow.  
LINT TIME: CPU TIME = 0 REAL TIME = 0
```

- **log_file_name *name* [-dump]**

An optional argument set specifying the path- and filename of the Calibre MDP summary report. Calibre issues a parsing error if you specify a directory that does not exist. The -dump option allows you to optionally echo the entire log file in the main Calibre fracture transcript. The log file is intended to provide a small archiveable summary of the

circumstances of the data preparation and its contents. It is also useful when loaded as a Calibre MDPview Overlay Alignment instruction. There, it will automatically re-orient the magnified, rotated, mirrored, scaled output pattern file to match the input data alignment and the Calibre MDPverify results for visual inspection.

The following is an example of the log file (information presented differs based on the Calibre FRACTURE format):

```
=====
=====
== CALIBRE::MDP SUMMARY REPORT
==
Running on <MACHINE INFORMATION>
Execution Date/Time:      <DATE/TIME INFORMATION>
Calibre Version:          <VERSION INFORMATION>

-----
--- GENERATION PARAMETERS
---
fracture region           (-440.000000, -335.000000) x (-200.000000, -
135.000000)
file_name                  TEST1XXXX.PF
maskshop_info
address_size                0.025
magnify                     4

rule file layer            orig6
-----

--- FORMAT FILE SUMMARY
---
mode :                      4
total segments:             2
total stripes per segment: 32
manhattan count             9237
parallelogram count         0
trapezoid1 count            32
trapezoid2 count            0
trapezoid3 count            0
compaction manhattan count  2120
compaction parallelogram count 0
compaction trapezoid1 count 0
compaction trapezoid2 count 0
compaction trapezoid3 count 0
```

- mirror {*x* | *y* | *x y* | *y x*}

An optional argument set defining the mirroring behavior to be applied to *layer* prior to fracturing.

- Mirroring occurs after the region is isolated.
- Mirroring is based around axes passing through the center of the region.

The axes must be the input data's coordinate system.

- rotate {0 | 90 | 180 | 270}

An optional argument set specifying the amount of rotation to be applied to *layer* prior to fracturing.

- Rotation occurs after the region is isolated and any mirroring transformation.
- Rotation is around the origin.

The origin must be the input data's coordinate system.

- shift x y

An optional argument set that shifts fractured data by the amount specified in x and y directions defined in units of microns at wafer scale. Fracture only supports positive shift. The amount of positive shift is limited by format restrictions on the output chip extent. This keyword can only be used in section mode fracture. The default is (0, 0).

Note

 This option is currently supported by Nuflare, Nuflare MBF, JEOL, and MEBES formats.

- magnify {1 | mag}

An optional argument set specifying the magnification to be applied to *layer* prior to fracturing. Setting this option to 1 specifies the default magnification. You can specify a magnification value by specifying a value for *mag*. Valid values are a positive integer or a fraction in the following forms:

```
magnify 4 //Magnify 4x
magnify 388/400 //Magnify 97% == 3% linear shrink
```

Tip

 The most common magnifications are 4 or 5, whichever is typical for the mask and wafer scale.

- reverse_tone

An optional keyword that causes the *layer* to be inverted within the fracturing region.

Note

 The default order of application of fracture transformation keywords is mirror, followed by rotate, then shift, and finally magnify. If reverse tone fracture is performed, then all the transformations are applied first in the described order and then reverse tone is applied before fracture.

- direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]

An optional argument set that permits remote processes in a Calibre MTflex invocation to directly access a file system. Direct file access enables better scaling for very large clusters using high-quality networks and disks in cases where the primary may become a bottleneck

in scaling. However, in order to use direct_FS_access, you must take precautions that the hardware configuration and path specifications comply with requirements:

- Specify full paths for the input, intermediate, and output paths in your rule files and remote configuration file (including LOCAL HOST DIR).
- Make sure that the file paths you specify are visible from the remotes when logged onto the remotes, and when using INTERMEDIATE.
- Make sure that the file system conforms to POSIX conventions described as follows:

INPUT — This keyword causes remote processes to read OASIS direct input files, including “adjunct data” such as injection files, via system calls rather than the Calibre Remote Protocol.

INTERMEDIATE — This keyword causes remote processes to write temporary intermediate fracture files directly to the file system via system calls rather than through the Calibre Remote Protocol. If INTERMEDIATE is specified, the file system must conform to POSIX conventions for file writing when “_POSIX_SYNCHRONIZED_IO” is defined. This avoids problems reading a file shortly after it is written by a different process.

OUTPUT — Some formats compute functions on the completed output files; for these cases, the OUTPUT argument enables direct read-access to the output files. If OUTPUT is specified, the output filename path must be an absolute path. If a relative path is specified, the absolute path is computed internally and used if the path is accessible by the remotes.

When remote processes directly access a file system, paths in that file system must be visible to those processes. This implies that when INPUT is used, the argument to vboasis_path must be visible to remote processes, and when INTERMEDIATE is used, the Calibre MTFlex LOCAL HOST DIR path must be visible to remotes; any argument to data_flow_branch must also be visible to remotes in these cases.

This keyword supports the use of parallel-distributed file systems. The default setting is that the option is not present.

- trapezoid_groups {*map_name annotation*}...

An optional argument set that enables multilayer trapezoiding, required for applications such as per-trapezoid shot rank annotation or per-trapezoid attribute information for dose-based MPC. This keyword requires the use of embedded SVRF to define the *map_name* and *annotation*. There must be a one-to-one matching from *map_names* to embedded SVRF output layer names. Each such layer is processed and has trapezoiding applied separately. The integer *annotation* may be used in particular ways by different fracture formats.

- JEOL — The annotation is transcribed to the “shot rank”; all trapezoids in a group have the shot rank whose value is the annotation for that group.
- OASIS.MASK — The annotation is transcribed to the data type; all trapezoids in a group have the data type whose value is the annotation for that group.

- VSB12i — The annotation is transcribed to Attribute Information; all trapezoids in a group have same Attribute Information (PH4 header) whose value is the annotation for that group.

The following illustrates a JEOL example of implementing trapezoid_groups with embedded SVRF:

```
MY_FRACTURE { FRACTURE JEOL poly FILE [
    file_name "mdp/m8051.jeol"
    log_file_name reports/m8051_jeol.log

    device JBX-3050MV
    version 3.1
    field_size 500
    fracture_units 2000
    fracture_units_2 1000
    field_oversize 2.0

    magnify 4

    small_value 0.1
    cd internal auto
    computation_mode section
    trapezoid_groups main 23 sbar 6

    <SVRFSTART>
    // Recognize SBARS.
    sbar = RECTANGLE poly < 0.100 BY < 1.00
    sbar { COPY sbar }
    // Find the main features as the complement.
    main { poly NOT sbar }
    <SVRFEND>
]
}
```

- trapezoid_pass_through

An optional keyword that instructs hierarchical fracture with OASIS direct-input to preserve input trapezoids to the greatest extent possible. Trapezoids are only reprocessed if they are not legally placed into a fractured file.

This technique is useful for preserving custom trapezoiding (for example, hand-drawn trapezoids), and for quickly pivoting from an already fractured file (OASIS.MASK) to another format.

Note

 **Embedded SVRF Options** — The following arguments define an embedded block of SVRF commands, which allows you to combine other mask data operations efficiently into the same section for processing. Embedded SVRF is discussed in greater detail in Chapter 3, “[Section-Based Processing, Calibre Embedded SVRF, and MDP EMBED](#)”.

- <SVRFSTART>

An optional keyword indicating the beginning of the embedded SVRF statement group. This is required only when using embedded SVRF.

- *svrf_statements*

Optional Embedded SVRF statements placed inside an <SVRFSTART> and <SVRFEND> block. These statements must obey the same rules as SVRF with the additional constraints:

- Lines cannot be longer than 1023 characters. Calibre truncates any SVRF line to 1023 characters if you exceed this value.
- No less than one output rule check with a single polygon output is required.
- No left- or right-brackets are allowed.

You must specify at least one rule check within the embedded SVRF statement group. The output of this rule check is what Calibre uses as the input to the FRACTURE operation, and must only be polygon data, not edge or error data. Refer to the *Calibre Verification User's Manual* for more information about rule checks.

You cannot specify left- or right-brackets within the embedded SVRF statement group. Therefore, any occurrence of brackets must be replaced as follows:

Left bracket ([)	<LB>
Right bracket (])	<RB>

The PRECISION settings of the embedded SVRF statements must match that of the Calibre rule file containing the FRACTURE statement.

- <SVRFEND>

An optional keyword indicating the end of the embedded SVRF statement group. This is required only when using embedded SVRF.

Note

 The left and right angle brackets (<>) for <SVRFSTART>, <SVRFEND>, <LB>, and <RB> are literal and required. SVRFSTART and SVRFEND must also always be capitalized.

- *svrf_range r*

An optional argument that specifies the buffer size for section processing. The default is determined automatically and is specified in user units at mask scale.

- *embedded_svrf_scale {WAFER | MASK}*

An optional argument set that specifies the scale at which embedded SVRF is applied.

- svrf_layer_name {*layer_name* | {*layer_name oasis_layer_number* [*oasis_layer_datatype*]...}}

An optional argument that specifies the names, numbers, and optional data types of the embedded SVRF layers corresponding to OASIS.MASK geometry. For a single layer, the *layer_name* only is used. For multiple layers, each name, corresponding layer number, and optional data type in the OASIS file must be specified. When OASIS.MASK input is used with HDB input, the PRECISION settings of the HDB and OASIS.MASK file must match. All layer names can be specified as arguments to a single svrf_layer_name command or can be specified using multiple commands to improve the readability of the rule file. For example:

```
svrf_layer_name <name1> <layer_number1> [layer_datatype] ...
svrf_layer_name <name2> <layer_number2> [layer_datatype] ...
```

Further information on using embedded SVRF can be found in Chapter 3, “[Section-Based Processing, Calibre Embedded SVRF, and MDP EMBED](#).”

Note

 **Direct OASIS Input** — The following keywords are related to a method of direct input of OASIS files, also referred to as OASIS DDE (Direct Data Entry). OASIS DDE can accept any OASIS file, but the performance is best with a restricted form of OASIS with a limited cell extent and hierarchy depth, which is referred to in this document as VBOASIS (Vector-Based OASIS).

- vboasis_path *filepath* [-noCheck] [-fastindex [*layer datatype*]]

An optional argument set that specifies a path to a single OASIS DDE file. By specifying the path of a single OASIS DDE file, you can bypass the Calibre nmDRC hierarchical database (HDB) constructor and instead take input directly from the OASIS file. Several restrictions and caveats apply to the use of this alternate input method:

- Compile-time checking — Limited compile-time extent checking is possible since the extent of the OASIS file is not known until it is parsed entirely. The extent is thoroughly checked at runtime.
- Distributed processing — For distributed processing (Calibre MTFlex) with 32-bit remotes, the maximum allowed data size of any cell in the OASIS file is 2GB, although the practical limit is determined by the remote memory. Files generated by conversion from any fracture format always have sufficiently small cells.
- OASIS file indices — Section mode will always attempt to reuse an existing index file for the input OASIS file. Any index constructed by on-demand in-batch processing will not contain viewer components, and so may not be suitable for use in the viewer.
- The PRECISION settings of the Calibre DRC hierarchical database and the vboasis_path OASIS input file must match.

While any valid OASIS DDE file that meets these restrictions can be used as an argument to this keyword, you must only use files with small abutting cells of shallow hierarchy, such as OASIS.MASK files, the outputs of the Calibre MDPview format-to-OASIS converters, or files created in Calibre with MDP CHECKMAP output. These files have hierarchies that can be effectively used as spatial indices. Using OASIS files not constrained by this trait is likely to result in degraded performance.

Tip

 For OASIS input, use of STRICT CBLOCK format is recommended.

Several side effects occur when input is taken from an OASIS DDE file. The HDB PRECISION setting is temporarily replaced by the PRECISION setting in the OASIS DDE file for the duration of the FRACTURE command. Also, the default HDB extent may no longer be a meaningful way of specifying the FRACTURE region, because there is no relationship between the HDB and input OASIS file (for instance, the EXTENT in INSIDE OF EXTENT refers only to the HDB extent, not the DDE extent).

An environment variable name preceded by a dollar sign (\$) can be used as an argument to vboasis_path. The variable is searched in the path specified by the environment variable, and its value is used as the filepath.

The OASIS index file is read or written as *file_path.fvi*. Thus, the location of the index file is the same as that of *filepath*. This behavior can be changed by specifying a colon-separated list of directories using the UNIX environment variable MDPIIndexSearchPath (refer to the section “Setting the Location of the Index File” in the *Calibre MDPview User’s and Reference Manual* for full details). The directories are searched in the order they are listed for reading or writing the index file.

The -noCheck option can be specified when checks on the input OASIS DDE layout file are not possible because it has not yet been generated. This would typically happen when the file is expected to be generated by another SVRF command and thus is not available during parsing for checking purpose. When using -noCheck, care should be given to specifying the file path and associated PRECISION values to avoid errors that are encountered much later.

To handle OASIS files with bad hierarchy (such as having redundant cells placed on top of each other), set the following environment variable:

```
CALIBRE_MDP_IGNORE_REDUNDANT_CELLS {true | false}
```

to true. All section-based processing ignores such redundant cells when this environment variable is enabled, in effect reducing memory usage.

The -fastIndex keyword enables a “fast mode” of index generation if the input file is an OASIS.MASK file. The file created in fast mode is valid only if the input OASIS.MASK file contains a single layer-datatype pair. Fast mode index generation should be avoided when there are multiple layers or datatypes in the input. By default, fast mode index generation assumes layer-datatype of 0-0. You can override these values by specifying layer-datatype pair as an argument to -fastIndex option.

Compressed layouts (those ending with a .gz extension) are not supported by vboasis_path.

- vboasis_precision_multiplier {*n[/d]* | AUTO}

An optional argument set that multiplies the PRECISION value by the supplied number *n*, and then scales the data from the file by the same amount to retain the same absolute scale as the original PRECISION setting. The default value for *n* is 1. The */d* is used to express a rational number as a fraction (for example, a multiplier of 1.5x would be declared as “3/2”).

There may be occasions when you want to re-interpret the PRECISION setting of the OASIS DDE file as specified using the vboasis_path keyword. For instance, you may want a higher resolution to allow finer shape modifications in embedded SVRF code.

If AUTO is supplied instead of a ratio, Calibre attempts to infer the correct ratio but fails if it is not a rational number, or if the combination of inferred numerator and denominator would cause arithmetic overflow. This keyword requires the use of embedded SVRF.

- vboasis_injection [-size *bin_size* | -size_factor *size_multiplier*] [-preserve 0 | 1] [-force] [-withIndex]

An optional argument set that specifies OASIS bin partitioning. The OASIS file specified using vboasis_path may contain small abutting cells of shallow hierarchy, but you can input a file that contains cells that have large amounts of data and have a large extent. These large cells degrade performance in section mode processing. To overcome this issue, cells can be partitioned into bins.

The size of the cells that must be binned and the size of each bin (*bin_size*) is determined automatically to a reasonable value vboasis_injection argument is specified. You can optimize the run time by controlling the bin size. This can be done by providing a size multiplier that acts as a multiplier on the automatically-determined size using the option -size_factor *size_multiplier*. The default value for *size_multiplier* is 1.0.

Alternately, the physical *bin_size* in microns can be specified explicitly using the option -size *bin_size*. Any cell whose extent has width or height more than *bin_size* is considered for binning.

The vboasis_injection option creates temporary data that may be reused if it is preserved using the -preserve option. The default is 0 (temporary data is not preserved). Use the -force option to ignore any existing data.

The temporary data is read or written in a directory named *file_path.lcb* where *file_path* is specified by vboasis_path. The location of this directory is the same as that of *file_path*. This behavior can be changed by specifying a colon-separated list of directories using the UNIX environment variable MDPIndexSearchPath (refer to the section “Setting the Location of the Index File” in the *Calibre MDPview User’s and Reference Manual* for full details). The directories are searched in the order they are listed for reading or writing the temporary directory.

For OASIS files with cell offsets, large cell binning can be done in parallel with the index generation using the -withIndex keyword. This option reduces the run time of index generation and binning combined. This keyword can only be used in conjunction with section-based fracturing (computation_mode section) and the size of each bin must be

explicitly specified using the `-size` keyword. If a valid index file is already present, then binning is done as a separate step.

- `section_count_limit limit`

An optional argument that increases the temporary file limit for Calibre FRACTURE. By default, any Calibre FRACTURE operation is limited to 1000 temporary files, except for VSB11, VSB12, VSB12i, NUFLARE_MBF, and JEOL where the default limit is 20,000. This directly limits the number of sections, or independent parallel tasks, that are defined in the section-processing part of the Calibre FRACTURE operation. You can override this setting for very large input layouts, complex embedded rule files, or large clusters. For these cases, the increased run time justifies a larger amount of parallel processing, but you must ensure the file system can tolerate the larger number of files. Specifically, the default value of 1024 for Linux[®]¹ maximum file descriptors should be raised by your system administrator to the largest value allowed for your system, to 64000. Once this is done, you can raise the value for `section_count_limit` to ~10% less than the maximum file descriptors. For VSB11, VSB12, VSB12i, and NUFLARE_MBF, the effective section count limit is the maximum of `section_count_limit` and the number of frames before frame splitting, if any. This keyword does not apply for OASIS.MAPPER. For OASIS.MAPPER, the effective section count is decided by the number of level-2 cells per stripe.

- `mdp_remote_tmp`

An optional keyword that sets the temporary directory for Calibre MTflex processes. A single path is used for all computers used by the invocation of Calibre. The path must be visible to all computers or run time exceptions will occur. The default value is `/tmp`.

Note



This directory should be on a file system local to each computer. If this is not done, it will introduce system performance issues.

1. Linux[®] is a registered trademark of Linus Torvalds in the U.S. and other countries.

Common Arguments for Vector Beam Formats

Supported for: [FRACTURE HITACHI \(FRACTUREh\)](#), [FRACTURE JEOL \(FRACTUREj\)](#), [FRACTURE NUFLARE \(FRACTUREt\)](#), [FRACTURE MICRONIC \(FRACTUREc\)](#), and [FRACTURE OASIS_MASK \(FRACTUREv\)](#)

There are several arguments that are common for vector beam FRACTURE formats only.

Usage

```
FRACTURE type layer [...layerN]
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]
[FILENAME output_file_name]
--> {FILE {parameter_block_name | '[' parameter_list '']}}
```

Arguments

- `small_value {value}`

An optional argument controlling the size of figures created by the FRACTURE operation.

`value` — specified in microns at mask scale. The default is 0.15 um (for Hitachi, JEOL and OASIS) and 0.1 um (for Micronic and NUFLARE).

This operation optimizes figure creation to avoid creating figures where the length of any edge is less than `value`.

- `cd {{internal {cd_max_width cd_min_length | auto}} | {external cd_min_width cd_max_width cd_min_length}}`

An optional argument specifying the maximum width and length criteria for critically dimensioned portions of the design. Critically-dimensioned design portions are horizontal or vertical features (or portions of features) that meet the following criteria:

- Have an external distance between them less than `cd_max_width` and greater than `cd_min_width`.
- Are longer than `cd_min_length`.

The following options are available:

- `internal {cd_max_width cd_min_length / auto}` — An “internal CD” describes critical dimensions within features on a mask.

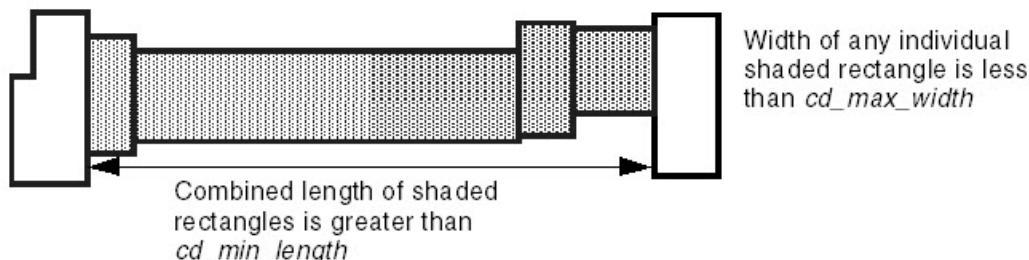
The internal keyword identifies critically dimensioned horizontal or vertical features (or portions of features) that are:

- Narrower than the `cd_max_width`
- Longer than the `cd_min_length` value.

If auto mode is specified, all polygons are captured, regardless of their width, and `cd_max_length` is automatically set to 1.5x `cd_max_width`. Calibre fractures any portion of the design that meets these criteria perpendicular to the length of the critically dimensioned portion, which results in better control of the CD on the mask.

This argument works across cell boundaries. This functionality allows jogs in a critically dimensioned portions, as long as the width of the portion is always less than *cd_max_width*. The following figure shows a critically dimensioned portion with jogs, represented by the shaded area. All of the shaded rectangles, which make up the portion, are less than *cd_max_width* wide. The length of the feature (all the shaded rectangles combined) is greater than *cd_min_length*.

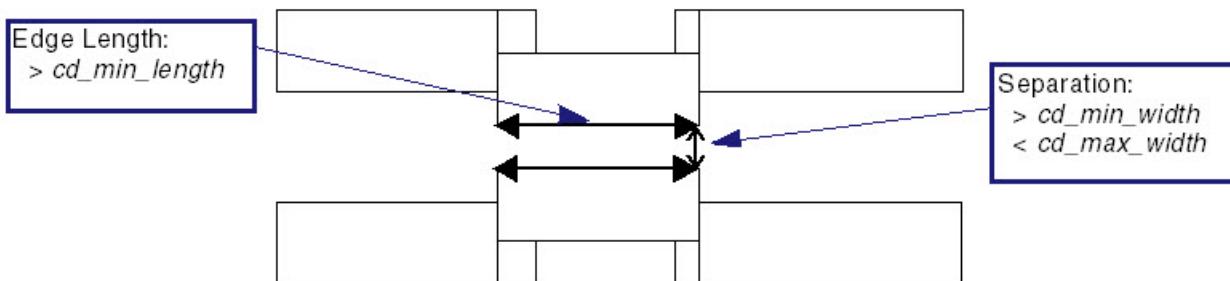
Figure 2-3. Internal Critically Dimensioned Portion of a Design



- external *cd_min_width* *cd_max_width* *cd_min_length* — An “external CD” describes a critically dimensioned feature on a reverse-tone mask. Using this optional keyword, the software converts data to trapezoids so that external CD edges belong to large trapezoids and are not broken. This functionality is useful for decreasing gate width variability on the mask for the case of reverse tone masks. Use of this option can increase the shot count and the number of small figures on the mask.

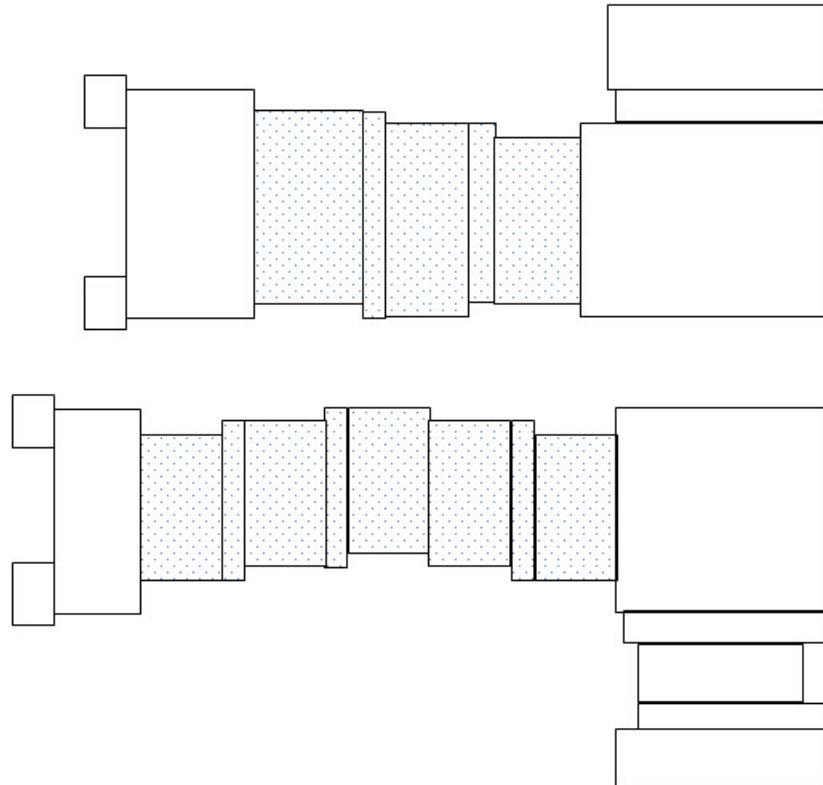
The following figure shows an example of an external CD. The highlighted edges identify an external CD that meets the requirements specified with *cd_min_length*, *cd_min_width*, and *cd_max_width*. Because this external CD exists, Calibre fractures the data such that these edges are not broken.

Figure 2-4. External Critically Dimensioned Feature



The shaded trapezoids in the following figure are considered critical in a normal tone fracture and illustrate trapezoiding style of the internal keyword.

Figure 2-5. Critical Features in a Normal Tone Fracture



The shaded trapezoids in the following are considered critical in a reverse tone fracture and illustrate trapezoiding style of the external keyword.

- `shot_size s`
An optional argument that defines the maximum size of a square shot. This argument is specified in microns at mask scale and controls trapezoiding. The allowed range is between 0.24 and 3.0 and the default value is 2.0. Customize this value to match the specification of the mask writer maximum `shot_size` argument in order to reduce shot count and improve mask writing time.
- `reduce_shot_count {1 | 2}`
An optional argument set that enables different shot count reduction methods in the Fracture flow. The supported methods are as follows:
 - 1 — This is the default setting.
 - 2 — Provides improved shot count reduction at the expense of increased run time.
- `reduce_run_time {0 | 1 | 2}`
An optional argument set that enables different run time reduction methods in FRACTURE operations.
 - 0 — No additional run time optimization is performed. This is the default setting.

- 1 — Reduces run time with no impact on the quality of results. This may also reduce the shot count.
- 2 — Reduces run time more aggressively with minor impact on the quality of results.

FRACTURE HITACHI (FRACTUREh)

SVRF mask data preparation command for Hitachi formats.

Usage

Calibre FRACTURE Syntax

```
FRACTURE HITACHI layer [...layerN]
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]
[FILENAME output_file_name]
{FILE {parameter_block_name | '['parameter_list'']} }
```

where *parameter_list* can contain the following arguments:

Global Calibre FRACTURE Arguments

```
file_name output_file_name
[computation_mode {hier | section}]
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[magnify mag]
[reverse_tone]
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]
[trapezoid_pass_through]
[trapezoid_groups {map_name annotation}...]
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}}...}]
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]]
[vboasis_precision_multiplier {n/d | AUTO}]
```

```
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]
[-preserve 0 | 1]]
[section_count_limit limit]
```

Common Arguments for Vector Beam Formats

```
[small_value value]
[cd {{internal {cd_max_width cd_min_length | auto}} |
{external cd_min_width cd_max_width cd_min_length}}]
[shot_size s]
[reduce_shot_count {1 | 2}]
[reduce_run_time {0 | 1 | 2}]
```

HITACHI-Specific Arguments

```
format_type {1 | 3 | 7}
max_skew_approximation_error skew_error //Required only for format_type {1 | 3}
lsb1 number
    //Required only for format_type 7
[b1 number]
[b2 number]
[b3 number]
[ssf_size size]
[ssf_margin size]
[comment text]
[large_output_file]
```

Arguments

- **Global and Common Arguments**

The HITACHI format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23
- “[Global Calibre FRACTURE Arguments](#)” on page 29
- “[Common Arguments for Vector Beam Formats](#)” on page 40

- **format_type {1 | 3 | 7}**

A required argument defining the format type for the Hitachi output file.

- 1 — Specifies format type for HL-800.
- 3 — Specifies format type for HL-900D and HL-950M.
- 7 — Specifies format type for HL-7000M.

Hitachi formats have a maximum file size, depending on the format_type settings. This is summarized in the following table:

Table 2-1. Hitachi Format Type Maximum File Size

Format	format_type Setting	Maximum File Size
HL-800M	1	<= 2GB
HL-900D	3	<= 8GB
HL-950M	3	<= 64GB
HL-7000M	7	<= 64GB

- **max_skew_approximation_error *skew_error***

A required argument that applies only when generating format_type 1 (HL-800) or format_type 3 (HL-900D and HL-950M) output. Calibre ignores this argument when generating format_type 7 (HL-7000M) output.

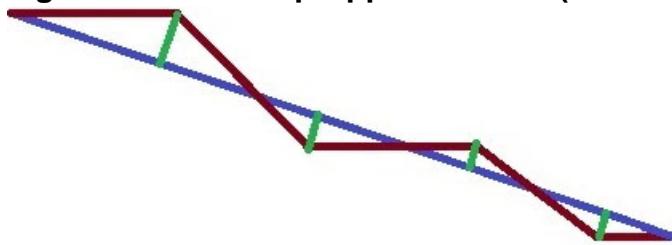
A required argument defining the maximum allowed distance between an approximation of a non-45 degree edge and the original edge.

skew_error — specified in microns at mask scale.

The tool approximates these edges with a staircase created from either horizontal and vertical lines (when close to 90 degrees) or horizontal, vertical, and 45-degree lines (when close to 45 degrees).

In [Figure 2-6](#), the green lines show the distance between the line and the approximation, which cannot exceed **max_skew_approximation_error**.

Figure 2-6. Stairstep Approximation (Hitachi)



- **lsb1 number**

This argument applies only when generating format_type 7 (HL-7000M) output.

A required argument specifying the LSB1 denominator.

number — Specifies an integer between 100 and 10,000, inclusive. It is the number of output units per micron with the LSB1 numerator currently fixed at 1.

Snapping could occur if this value is not an even multiple of the input PRECISION setting.

- **lsb2 number**

An optional argument specifying the LSB2 denominator.

number (format_type 1 and format_type 3) — Specifies an integer, specified in even multiples of the input PRECISION setting to prevent snapping errors. The default value is 1600.

number (format_type 7) — an integer, specified in even multiples of the input PRECISION setting to prevent snapping errors. The minimum value is 100 and the default value is the setting for **lsb1**.

- **lsb3 number**

An optional argument that applies only when generating format_type 7 (HL-7000M) output. This argument specifies the LSB3 denominator.

number — an integer, specified in even multiples of **lsb1** to prevent snapping errors. The default value is the setting for **lsb1**.

- **ssf_size size**

An optional argument specifying the sub-subfield size.

size (format_type 1 and format_type 3) — Specified in microns. The maximum value is 81.91 and the default is 60.

size (format_type 7) — Specified in microns. The maximum value is 1000 and the default is 900.

- **ssf_margin size**

An optional argument specifying the sub-subfield margin used in the sub-subfield margin division.

size (all format_type values) — Specified in microns at mask scale. The default is 0.5 *u*.

This option is used to reduce small figures at the sub-subfield boundary. Sub-subfield overlap is equal to 2*ssf_margin.

- **comment text**

An optional argument specifying text information to be inserted in the Hitachi control information record.

text — Specifies an alpha-numeric string that must be less than or equal to 63 characters.

- **large_output_file**

This argument applies only when generating format_type 3 (HL-900D and HL-950M) output.

An optional keyword specifying that you expect the output pattern file size to be larger than 8 gigabytes.

If you specify this keyword and the output file is smaller than 8 gigabytes, the output data is correct, but the file is larger than necessary.

If you do not specify this keyword and the output pattern data file size is larger than 8 gigabytes, the fracture operation issues an error and terminates processing.

Description

There are a number of notes that apply to FRACTURE HITACHI.

- Hitachi output filenames specified by the **file_name** global argument have certain restrictions on allowed extensions, as documented in [Table 2-2](#).

Table 2-2. Output File Name Restrictions for Hitachi

format_type	Machine	Allowed Extension
1	HL-800	.pfh .PFH
3	HL-900D or HL-950M	.pfh3 .PFH3
7	HL-7000M	.iph .IPH

- For FRACTUREh, the unit size is fixed for both HL-800 (10 nm) and HL-900 (2.5 nm), however for HL-7000 you control this setting with the lsb1 argument, which can be as low as 0.1 nm.

Tip

 By default, the FRACTURE HITACHI operation outputs the pattern file origin at the center of the fractured file. Use an even number of units for your input fracture region to balance the pattern extents around the center. When specifying an odd number of unit sizes, an extra address unit is added to the top and/or right side of the output pattern window (as required for a center placement).

- For FRACTUREh, the fracture operation outputs a single Hitachi pattern file with the path and filename as specified by the FILENAME keyword to the Fracture SVRF statement or the **file_name** in-line keyword.
- For a Hitachi pattern file, the filename portion of the pathname can contain alpha-numeric characters and the underscore (_) character. You can specify a period (.) only once as the delimiter for the extension field. The valid extensions are as follows:
 - HL-800 — .PFH or .pfh
 - HL-900 — .PFH3 or .pfh3
 - HL-7000 — .IPH or .iph
- For a Hitachi pattern file, the origin is located at the center of the chip.

FRACTURE JEOL (FRACTUREj)

SVRF mask data preparation command for JEOL formats.

Usage

Calibre FRACTURE Syntax

```
FRACTURE JEOL layer [...layerN]  
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]  
[FILENAME output_file_name]  
{FILE {parameter_block_name | '['parameter_list'']} }
```

where *parameter_list* can contain the following arguments:

Global Calibre FRACTURE Arguments

```
file_name output_file_name  
[computation_mode {hier | section}]  
[input_region_severity {0 | 1}]  
[index_options [-topcell cellname]]  
[lint]  
[log_file_name name [-dump]]  
[mirror {x | y | x y | y x}]  
[rotate {0 | 90 | 180 | 270}]  
[shift x y]  
[magnify mag]  
[reverse_tone]  
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]  
[trapezoid_pass_through]  
[trapezoid_groups {map_name annotation}...]  
<SVRFSTART>  
    svrf_statements  
<SVRFEND>]  
[svrf_range r]  
[embedded_svrf_scale {WAFER | MASK}  
[svrf_layer_name {layer_name | {layer_name oasis_layer_number  
[oasis_layer_datatype]}}...}]]  
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]]  
[vboasis_precision_multiplier {n/d | AUTO}]
```

```
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]
[-preserve 0 | 1]]
[section_count_limit limit]
```

Common Arguments for Vector Beam Formats

```
[small_value value]
[cd {{internal {cd_max_width cd_min_length | auto}} |
{external cd_min_width cd_max_width cd_min_length}}]
[shot_size s]
[reduce_shot_count {1 | 2}]
[reduce_run_time {0 | 1 | 2}]
```

JEOL-Specific Arguments

```
[device device_name]
[version version_number]
fracture_units gridsize
[fracture_units_2 units]
[field_size size | {field_size_x x_extent field_size_y y_extent}]
[field_oversize extension]
[fields_per_section fps]
[lb_limit limit_value]
[hsc_limit command_count_limit]
[exposed_area]
[trapezoid_groups {{map_name annotation}... | -doseMap {map_name dose}...}]
```

Arguments

- **Global and Common Arguments**

The JEOL format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23
- “[Global Calibre FRACTURE Arguments](#)” on page 29
- “[Common Arguments for Vector Beam Formats](#)” on page 40

- **device *device_name***

An optional argument specifying the intended device for the JEOL output file. Some arguments, such as field size and version, have limits that depend on the device type.

device_name — Specifies one of the following:

JBX-9000MV — This is the default.

JBX-9300FS-50kV

JBX-9300FS-100kV

JBX-3030MV

JBX-3040MV

JBX-3050MV

JBX-3200MV

- version {3.0 | 3.1}

An optional argument defining format version for the JEOL output file.

3.0 — This is the default value.

3.1 — This version can only be used with the JBX-3030MV, JBX-3040MV, and JBX-3050MV devices.

- **fracture_units gridsize**

A required argument specifying the number of output units per micron for the JEOL output file.

gridsize — an integer that satisfies the equation:

$$\textit{gridsize} = 1000/x$$

where *x* is the desired grid size in microns, for example:

Table 2-3. Grid Size Settings

Desired Grid Size	<i>gridsize</i>
2 nm	250
4 nm	250

Note

 Because *gridsize* must be an integer, the desired unit size (in nanometers) must evenly divide 1000.

For version 3.0, the values 2000, 4000, and 8000 are also allowed. For the JBX-3050MV, 2000 must be used for either format version (3.0 or 3.1).

- **fracture_units_2 units**

An optional argument that specifies the common PRECISION setting for the JEOL “unit of position-set” and “unit of chip size.” For JBX-3050MV and JBX-3200MV only, this field may have a value different from **fracture_units**. The only allowed combination of settings for the two arguments that are not the same is 2000 and 1000 for **fracture_units** and **fracture_units_2**, respectively. When the values are different, the **fracture_units** argument is only used for the JEOL “unit of pattern data.” Using different PRECISION settings makes possible finely-discretized data with a large field size. In fact, the mixed setting is required for using a field larger than 500um on the JBX-3050MV, and larger than 800um on the JBX-3200MV. The default value is the value of **fracture_units**.

- *field_size size | {field_size_x x_extent field_size_y y_extent}*

An optional argument specifying the field size for the mask writer. Choose one of the following:

field_size size — specifies the length of a side of a square JEOL format field.

- *size* (JBX-9300FS-100kV) — Specified in microns, the default value is 500, and the maximum value is 500.
- *size* (JBX-3050MV) — Specified in microns, if PRECISION 2000 is used, the maximum field size in either dimension is 500.
- *size* (all other devices) — Specified in microns, the default value is 1000, and the maximum value is 1000.

field_size_x x_extent field_size_y y_extent — specifies the length of the *x* and *y* sides of a rectangular JEOL format field. You must specify these arguments on two separate lines.

- *x_extent y_extent* (JBX-9300FS-100kV) — Specified in microns, the default value is 500 and the maximum value is 500.
- *x_extent y_extent* (all other devices) — Specified in microns, the default value is 1000 and the maximum value is 1000.

- *field_oversize extension*

An optional argument specifying how far the JEOL field can be extended beyond the right and bottom sides. This extension enables the use of a “soft boundary” at field edges in order to mitigate the creation of small figures at the boundaries. This value is specified in microns at mask scale. It is restricted to the range [0, 4.0] in microns. If not otherwise specified, the default value is 0.5 microns.

- *fields_per_section fps*

An optional argument that specifies the number of fields included per section. This argument takes effect only if a non-zero value is specified. If this keyword is not specified, then the number of fields in a section is dynamically computed based on *section_count_limit*, with a minimum of two fields per section.

- *lb_limit limit_value*

An optional argument that enables the user to limit the number of trapezoids written to Library Block (LB) records.

limit_value — An integer specifying the maximum number of trapezoids. A warning is written to the transcript if this limit is reached. The default value depends on the device:

JBX-3030MV and JBX-3040MV — 6,000,000

JBX-3050MV — 27,000,000

All other devices — 96,000,000

The total number of LB records is limited to:

JBX-3030MV — 32,702

JBX-3040MV — 65,470

All other devices — 1,000,000

- `hsc_limit command_count_limit`

If the sum of JEOL commands in all LB records and TX records for any single field exceeds this amount, a warning is issued for each violating field. The presence of this exception indicates that the “HSC memory overflow” problem is likely to arise on the JEOL writer when writing the resulting JEOL file. The default value is 8000000.

- `exposed_area`

An optional keyword that directs the program to compute and print the area of the pattern file that is covered by trapezoids. The covered area is printed as a percentage of the fracture region area and in square microns. The information is printed to the Calibre transcript.

- `trapezoid_groups {{map_name annotation}... | -doseMap {map_name dose}...}`

An optional keyword that enables multilayer trapezoidal groups. This functionality requires the use of embedded SVRF. There must be a one-to-one match of map_names to embedded SVRF output layer names. Each of these layers are processed and converted to trapezoids separately. The integer annotation can be used in various ways by different fracture formats:

- JEOL — The annotation is transcribed to a “shot rank”. All trapezoids in a group have a shot rank whose value is in the annotation for that group.

Specifying -doseMap indicates dose values are used instead of annotations. The dose value must be a number in the range of 0.0 and 2.0. If a value of 1.0 is specified in the list of dose values, the maximum number of dose values that can be specified is eight; otherwise, the maximum number is seven. The annotations are assigned internally as follows:

```
trapezoid_groups -doseMap L1 0.8 L2 0.75 L3 1.0 L4 1 .1
                  L5 1 .0 L6 1.2 L7 1.3     L8 1.4
```

The annotations for this specification are:

L1	0.8	Annotation	1
L2	0.75	Annotation	2
L3	1.0	Annotation	0
L4	1.1	Annotation	3
L5	1.0	Annotation	4
L6	1.2	Annotation	5
L7	1.3	Annotation	6
L8	1.4	Annotation	7

If the dose value list does not have a value of 1.0 specified, the annotation values are assigned integer values sequentially from 1 to n , where n is the number of dose values specified. Regardless of whatever value list is specified in `trapezoid_groups`, the standard dose value of 1.0 is always assumed to be present with its annotation set to 0.

Description

There are a number of notes that apply to FRACTURE JEOL.

- For Calibre FRACTUREj, you specify the unit size in terms of the **fracture_units**, which is expressed as (1/grid size in microns).
- For Calibre FRACTUREj, the FRACTURE operation outputs a single JEOL pattern file with the path and filename as specified by the FILENAME keyword to the Fracture SVRF statement or the **file_name** inline keyword.
- For a JEOL pattern file, the filename portion of pathname can contain up to 24 letters, numbers and symbols. Apostrophes ('') cannot be used.
- For a JEOL pattern file, the origin is always located at the upper left corner.

FRACTURE MEBES (FRACTUREm)

SVRF mask data preparation command for MEBES formats.

Usage

Calibre FRACTURE Syntax

```
FRACTURE MEBES layer [...layerN]
  [INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]
  [FILENAME output_file_name]
  {FILE {parameter_block_name | '['parameter_list '']}}
```

where *parameter_list* can contain the following arguments:

Global Calibre FRACTURE Arguments

```
file_name output_file_name
  [computation_mode {hier | section}]
  [input_region_severity {0 | 1}]
  [index_options [-topcell cellname]]
  [lint]
  [log_file_name name [-dump]]
  [mirror {x | y | x y | y x}]
  [rotate {0 | 90 | 180 | 270}]
  [shift x y]
  [magnify mag]
  [reverse_tone]
  [direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]]
  [trapezoid_pass_through]
  [trapezoid_groups {map_name annotation}...]
  [<SVRFSTART>
    svrf_statements
  <SVRFEND>]
  [svrf_range r]
  [embedded_svrf_scale {WAFER | MASK}]
  [svrf_layer_name {layer_name | {layer_name oasis_layer_number
    [oasis_layer_datatype]}}...}]
  [vboasis_path filepath [-noCheck] [-fastindex [layer datatype]])
  [vboasis_precision_multiplier {n/d | AUTO}]
```

```
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]
[-preserve 0 | 1]]
[section_count_limit limit]
```

MEBES-Specific Arguments

```
mode {4 | 5}
[auto_mode_switch]
address_size unit_size
[compaction_stripes {1 | 32 | 64}]
[mask_info text]
[angle_fix_yes]
[disk_file_name dname]
[generate_index_file]
[density [-suppress_from_header]]
```

Arguments

- **Global and Common Arguments**

The MEBES format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23
- “[Global Calibre FRACTURE Arguments](#)” on page 29

- **mode {4 | 5}**

A required argument defining the mode number for the MEBES output file.

- **auto_mode_switch**

The presence of this keyword enables the automatic mode switch from mode 4 to mode 5 if either the number of segments for MEBES data exceeds the mode 4 limit of 255, or the number of stripes per segment for MEBES data exceeds the mode 4 limit of 65534.

In other words, during a MEBES mode 4 run, if the auto_mode_switch is enabled and the number of MEBES segments or stripes exceeds the specified limit, the run automatically switches from mode 4 to mode 5. If the auto_mode_switch is not enabled and the number of MEBES segments or stripes exceeds the limit for a mode 4 run, the run exits.

- **address_size *unit_size***

A required argument specifying the unit size for the MEBES output file.

unit_size — A number, specified in microns. For mode 4 output, this value is limited to numbers between .025 *u* and 1.1*u*, inclusive. If the MEBES mode 4 address size falls outside the design space, the fracture operation completes with a warning and an MDP status message flags at the end:

“WARNING: 'address_size' should be in the range [0.025, 1.1] for MEBES mode 4”

- **compaction_stripes {1 | 32 | 64}**

An optional argument defining the stripe compaction to use.

- 1 — Default for mode 4.
- 32 — Does not apply to mode 4.
- 64 — Does not apply to mode 4 and is the default for mode 5.

When you use mode 4, setting this keyword to any value other than 1 results in an error that aborts processing.

- **mask_info *text***

An optional argument supplying text information for the photomask process.

text — a string of alpha-numeric characters that must be less than or equal to 40 characters.

The Calibre FRACTUREm software inserts this information into the header of each MEBES pattern file it generates for the layer.

This data is not used by the Calibre FRACTUREm software or the Calibre nmDRC hierarchical engine.

- **angle_fix_yes**

An optional argument activating preferential snapping to 45-degree angles. You must use this switch to maintain 45-degree angles on output, and only in conjunction with section-based fracturing (computation_mode section).

- **disk_file_name *dname***

Specifies that the output MEBES pattern data is written to the file with name specified by this *dname* instead of the one specified with the **file_name** keyword. This allows you to assign a name to the MEBES files of up 20 characters with underscores that would otherwise violate the file naming convention.

The disk_file_name argument must be used in conjunction with the **file_name** argument. The **file_name** argument assigns an internal name to the pattern that complies with the MEBES file naming convention (compliant with mode 4 and mode 5 MEBES files). For example:

```
disk_file_name testxxxxx_10s3.pf      //Optional name on disk
file_name testxxx10.s3                 //Required MEBES-compliant
internal name
```

If a disk_file_name is not specified, then the output MEBES pattern filename is specified by the **file_name** keyword. However, the pattern filename field in the MEBES header always stores the filename specified with the **file_name** keyword in either case.

- **generate_index_file**

This keyword can only be used when computation_mode is set to “section”. The presence of this keyword enables the generation of a MEBES stripe offset index file during MEBES fracturing, but without any viewer components that would aid viewer performance.

The generated index file is named by adding “.xvi” to the **file_name**. This index file helps to speed up Calibre MDPverify verification of large MEBES data files. Prior versions of

MEBES fracture generated an “.evi” for mode 5 only when computation_mode was set to section. To generate the “.evi,” set the UNIX environment variable MDP_MEBES_READER_VERSION_VALUE to 1 (the default value is 2).

- density [-suppress_from_header]

If this keyword is present, then the transmission density is computed and the following string is appended to the MEBES file header:

```
density value%
```

where *value* is the percentage of the transmission area relative to the overall area of the input layer extent, computed up to two decimal places rounded after the period. For example:

```
density 20.72%
```

The density value is also written to the log file, computed up to 4 decimal places rounded after the period. For example:

```
Transmission density = 12.3096%
```

If -suppress_from_header is specified, the computed density is not written to the MEBES header, and is used for reporting instead.

Description

There are a number of notes that apply to FRACTURE MEBES.

- For FRACTUREm, specify the unit size in terms of the *address_size*, which is written in microns.
- The unit size strongly influences coordinate snapping. When working with the MEBES format, which allows more flexibility with respect to the unit size, you can avoid introducing snapping-related errors by setting the unit size equal to a multiple of the input file PRECISION setting and a factor of the final design grid size. When the MEBES unit size is determined by the LITHO grids used for OPC, you can avoid data snapping between the design grid and the mask writer grid, by setting the *address_size* to the following:
 - Equal to or an integer multiple of the LITHO gridsize for the design.
 - Equal to or an integer factor of the LITHO stepsize for the design. If the data is magnified after LITHO, then address_size must be equal to or an integer factor of (LITHO stepsize * magnification).
- Calibre FRACTUREm outputs a single MEBES pattern file with the path and filename as specified by the FILENAME keyword to the Fracture SVRF statement, the **file_name** inline keyword, or disk_file_name.
- The filename portion of the path must be 12 characters and cannot contain any blanks or dashes. The last three characters define the file extension; the dot (.) plus a two-character extension.

- For a MEBES pattern file, the origin is always located at the lower left corner.
- Direct data entry (DDE) is not supported when performing a FRACTURE in MEBES hierachal mode (computation_mode hier). This produces an empty fracture file.
- The MEBES input module has two implementations which can be controlled by the UNIX environment variable MDP_MEBES_READER_VERSION_VALUE.

```
setenv MDP_MEBES_READER_VERSION_VALUE [1 | 2]
```

The default value 2 uses a newer MEBES input module that reads a stripe offset index file named “*.xvi” to enhance performance.

Setting the variable to 1 uses an older version of the MEBES input module, which reads an index file named “*.evi” only for mode 5.

- The MEBES header is limited to 40 characters. Both the mask_info and density command write information to the MEBES header by default. Depending on whether you specify mask_info, density, or both, the header information can be formatted differently to adhere to the 40 character limit.
 - If you specify both mask_info and density, then the information for both are concatenated together. For example, if mask_info is less than 25 characters, such as “1234567890123456789012345”, and density is “density 006.10%”, the output is “1234567890123456789012345density 006.10%”. However, if mask_info is greater than 25 characters, the job fails at compile time.
 - If you specify only mask_info with less than 40 characters (for example, “mask_info 123456789012345678901234567890”), the output is 1234567890123456789012345678901234567890. If you just specify “mask_info”, with mask_info argument greater than 40 characters, the job fails at compile time.
- The MEBES format currently limits the number of segments at 10000. Because address units below 0.0005 requires more than this limit, Calibre MDP extends the limit to 50000 segments using the following environment variable:

```
setenv CALIBRE_RELAX_MEBES_SEGMENT_LIMIT true
```

FRACTURE MICRONIC (FRACTUREc)

SVRF mask data preparation command for Micronic formats.

Usage

Calibre FRACTURE Syntax

```
FRACTURE MICRONIC layer [...layerN]
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]
[FILENAME output_file_name]
{FILE {parameter_block_name | '['parameter_list '']}}
```

where *parameter_list* can contain the following arguments:

Global Calibre FRACTURE Arguments

```
file_name output_file_name
[computation_mode {hier | section}]
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[magnify mag]
[reverse_tone]
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]
[trapezoid_pass_through]
[trapezoid_groups {map_name annotation}...]
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}}...}]
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]}}
  [vboasis_precision_multiplier {n/d | AUTO}]
```

```
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]
[-preserve 0 | 1]]
[section_count_limit limit]
```

Common Arguments for Vector Beam Formats

```
[small_value value]
[cd {{internal {cd_max_width cd_min_length | auto}} |
{external cd_min_width cd_max_width cd_min_length}}]
[shot_size s]
[reduce_shot_count {1 | 2}]
[reduce_run_time {0 | 1 | 2}]
```

MICRONIC-Specific Arguments

```
version {1.6 | 1.8}
[scale [+integer | +float | +int_numer +int_denom]
[shift_origin_ll {0 | 1}]
[runtime_exception_severity file_name_restrictions [-1 | 0 | 1]]
[comment text]
[enable_checksum]
[density wx wy file_name]
```

Arguments

- **Global and Common Arguments**

The MICRONIC format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23
- “[Global Calibre FRACTURE Arguments](#)” on page 29
- “[Common Arguments for Vector Beam Formats](#)” on page 40

- **version {1.6 | 1.8}**

A required argument specifying the version of the Micronic format. If you do not specify a version, an error message is generated.

- **scale [+*integer* | +*float* | +*int_numer* +*int_denom*]**

An optional argument that specifies the data unit for coordinates, specified in nanometers. The value can be a range between 10⁻⁴ and 10⁺⁶. The value can be a single number (integer or float), or a pair of integers that define a ratio (numerator/denominator). The default for this option is 1. Any value specified must divide evenly by 1000.

- **shift_origin_ll [0 | 1]**

An optional argument that shifts the origin of the output coordinates in the fracture file.

1 (default) — The lower-left corner of the fracture area is shifted to the origin on output.

0 — The origin is kept at its current location.

- `runtime_exception_severity` `file_name_restrictions` [-1 | 0 | 1]

An optional argument that determines the behavior of MICRONIC fracturing operation if it encounters an output filename which does not adhere to the Micronic output naming conventions.

-1 — The file naming convention is ignored. This is the default setting.

0 — The fracturing operation continues, even though the output filename violates the convention. In this case, the following warning message is generated and appears in the transcript.

WARNING: MDP exit status: 32

1 — The fracturing operation immediately exits upon encountering a non-conforming output filename.

The MICRONIC output filename convention is:

- The basic syntax is *filename.extension*.
- The *filename* contains 1 to 12 characters from the following set:
 - upper-case letters “A” through “Z”
 - lower-case letters “a” through “z”
 - digits “0” through “9”
 - underscore (_)
- The *extension* is either “la” or “LA”.

For example, the following names are valid:

```
MIC_file.la  
upto_12chars.LA
```

The following names are invalid:

```
abc.MIC  
abc.def.la
```

- `comment` *text*

An optional argument specifying text information to be inserted in the file header.

text — an alpha-numeric string that must be less than or equal to 79 characters.

Double-quotes (“ ”) are required only for an empty string (default), and are optional for strings that consist of at least one character and no spaces.

- `enable_checksum`

An optional keyword that triggers the computation of section-order-independent checksum of the output file. If specified, the checksum is written to the output transcript and the log file in the following form:

OUTPUT CHECKSUM: *value*

This option is deactivated by default.

- `density wx wy file_name`

An optional argument that produces a density map according to Micronic specifications (Micronic document aa36182). The three required arguments describe the window size and density filename. Arguments *wx* and *wy* are the window width and height, respectively, in microns at mask scale. The window sizes must evenly divide the corresponding fracture field size, 500umx500um. Density information may only be generated in the section fracture mode.

Note



The density window cannot be larger than the fracture field size and needs to evenly divide the field size.

Description

There are a number of notes that apply to FRACTURE MICRONIC.

- Use the FRACTURE MICRONIC scale option to specify how many nanometers (or divisions of nanometers) are in the output file. The default size is a nanometer.
- For a MICRONIC pattern file, the filename portion of the pathname can contain alphanumeric characters and the underscore (_) character. You can specify a period (.) only once as the delimiter for the .la extension.
- For a MICRONIC pattern file, by default, the origin is automatically moved to the lower left corner of the fracture area. However, you have the ability to keep the origin where it is in the input.

FRACTURE MICRONIC and Origin Shifting

The FRACTURE MICRONIC operation has a keyword, `shift_origin_ll`, that allows you to shift the output coordinates automatically to the lower left corner of the fracture area. It has two settings: if set to 1 (the default), the output coordinates are automatically shifted to the lower-left corner; if set to 0, the origin is kept at its current GDS or OASIS location. However, Calibre always treats the bottom-left corner as the origin regardless where it actually is. Though in Micronic you can choose where the origin is by setting `shift_origin_ll`, either choice does not influence how Calibre MDPverify aligns the data.

Figure 2-7 and Figure 2-8 illustrates an example:

Figure 2-7. Micronic Origin Shift

In this example, two identical regions of Layer 1 are fractured for Micronic format, using different settings for shift_origin_ll.

FRACTURE Code - shift_origin_ll Set to 0

```
micronic_16 {
    fracture micronic Layer1 \
        inside of -1 3 50 50 file [
            file_name micronic_16.la
            log_file_name micronic_16.log
            version 1.6
            magnify 4
            scale 2
            shift_origin_ll 0
        ]
}
```

FRACTURE Code - shift_origin_ll Set to 1

```
micronic_16_b {
    fracture micronic Layer1 \
        inside of -1 3 50 50 file [
            file_name micronic_16b.la
            log_file_name micronic_16b.log
            version 1.6
            magnify 4
            scale 2
            shift_origin_ll 1
        ]
}
```

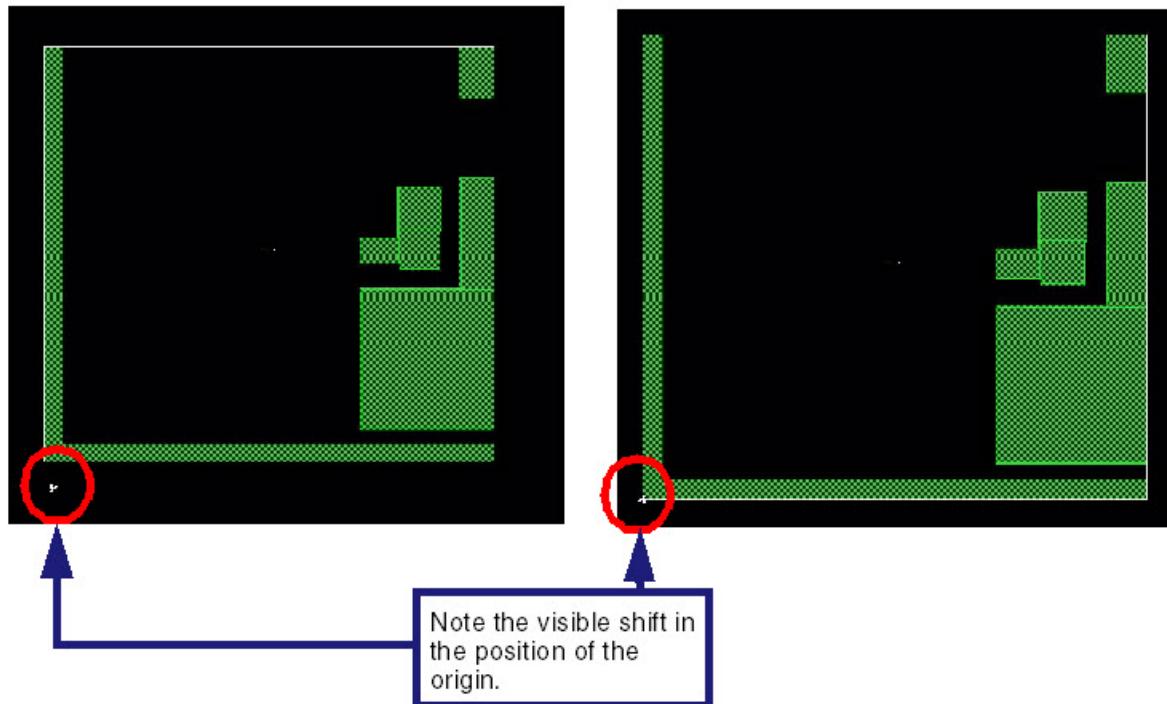


Figure 2-8. Micronic Origin Shift Continued

However, in Calibre MDPverify, the bottom-left corner is always treated as the origin, regardless of where it actually is. Thus, the Calibre MDPverify code would be the same for both cases.

Calibre MDPverify Code for Database 1

```
xor1 { mdpverify Layer1 \
inside of -1 3 50 50 file [
    input_file mebesmod5.pf \
        micronic_16.la
    verify_type mebes2micronic
    magnify 4 4
    mirror1 x y
    shift -1 3 -1 3
]
```

Calibre MDPverify Code for Database 2

```
xor16b { mdpverify Layer1 \
inside of -1 3 50 50 file [
    input_file mebesmod5.pf \
        micronic_16b.la
    verify_type mebes2micronic
    magnify 4 4
    mirror1 x y
    shift -1 3 -1 3
]
```

FRACTURE NUFLARE (FRACTUREt)

SVRF mask data preparation command for Nuflare formats.

There are several variants: [VSB11 Format](#), [VSB12 V1 Format](#), [VSB12 V2 Format](#), [VSB12i Format](#).

VSB11 Format	67
VSB12 V1 Format	73
VSB12i Format	80
VSB12 V2 Format	88

VSB11 Format

The VSB11 format for FRACTURE NUFLARE has its own specific arguments.

Usage

[Calibre FRACTURE Syntax](#)

FRACTURE NUFLARE *layer* [...*layerN*]
[INSIDE OF {EXTENT | *x1 y1 x2 y2* | LAYER *layer2*}]
[FILENAME *output_file_name*]
{FILE {*parameter_block_name* | '['*parameter_list* '']}}

where *parameter_list* can contain the following arguments:

[Global Calibre FRACTURE Arguments](#)

[computation_mode {hier | section}]
[input_region_severity {0 | 1}]
[index_options [-topcell *cellname*]]
[lint]
[log_file_name *name* [-dump]]
[mirror {*x* | *y* | *x y* | *y x*}]
[rotate {0 | 90 | 180 | 270}]
[shift *x y*]
[magnify *mag*]
[reverse_tone]
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]
[trapezoid_pass_through]
[trapezoid_groups {*map_name annotation*}...]
<SVRFSTART>
 svrf_statements
<SVRFEND>]
[svrf_range *r*]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {*layer_name* | {*layer_name oasis_layer_number*
 oasis_layer_datatype}...}]]
[vboasis_path *filepath* [-noCheck] [-fastindex [*layer datatype*]])]
[vboasis_precision_multiplier {*n/d* | AUTO}]

```
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]
[-preserve 0 | 1]]
[section_count_limit limit]
```

Common Arguments for Vector Beam Formats

```
[small_value value]
[cd {{internal {cd_max_width cd_min_length | auto}} |
{external cd_min_width cd_max_width cd_min_length}}]
[shot_size s]
[reduce_shot_count {1 | 2}]
[reduce_run_time {0 | 1 | 2}]
```

VSB11-Specific Arguments

version 11

```
chip_directory output_directory
max_skew_approximation_error skew_error
[conversion_address_unit {0.00125 | unit}]
[frame_width {512 | width}]
[cell_max_height {1024 | mheight}]
[cell_max_width {128 | mwidth}]
[cell_subfield_size {64 | size}]
[cell_subfield_margin {0.25 | size}]
[frame_max_data {256 | fdata}]
[common_max_data {256 | cdata}]
[frame_bde_max_data {256 | fbdedata}]
[common_bde_max_data {256 | cbdedata}]
[runtime_exception_severity bde_filesize_limit [0 | 1]]
[runtime_exception_severity frame_width_range [0 | 1]]
[pattern_array_compression {0 | 1}]
[pattern_compression {0 | 1}]
[pattern_set description]
[chip_name name]
[make_chip_ini]
[array_cell_cost_multiplier {4 | multiplier}]
[exposed_area]
```

Arguments

- **Global and Common Arguments**

The VSB11 format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23
- “[Global Calibre FRACTURE Arguments](#)” on page 29

- “[Common Arguments for Vector Beam Formats](#)” on page 40
- **version 11**
A required argument specifying VSB11.
- **chip_directory *output_directory***
A required argument specifying the directory path in which to store the fractured data.
Calibre creates the specified directory if it does not exist. For VSB11, the directory is similar in structure to:

```
<output_directory>/  
    common.chp  
    frame.<x>  
    chip.cnf
```

Calibre overwrites the specified directory at the time of execution if it already exists. Environment variables can be used in the path specification of the ***output_directory***.

This keyword is mutually exclusive from the FILENAME or **file_name** keywords; Calibre generates an error if you specify both for any one fracture operation.

Tip

 Use this argument to specify the directory of the fracture output if you specify all the other arguments inline to the fracture operation.

- **max_skew_approximation_error *skew_error***

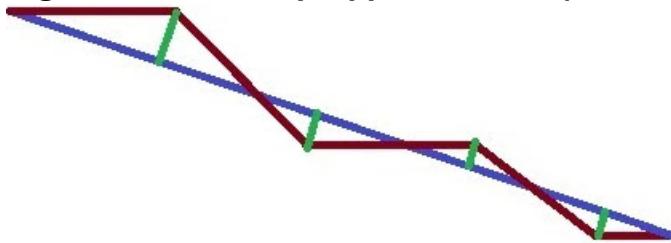
A required argument defining the maximum allowed distance between an approximation of a non-45-degree edge and the original edge.

skew_error — specified in microns at mask scale. The value must be greater than or equal to 1 conversion address unit.

The tool approximates these edges with a staircase created from either horizontal and vertical lines (when close to 90 degrees) or horizontal, vertical, and 45-degree lines (when close to 45 degrees). Since VSB11 only allows trapezoids with angles that are multiples of 45 degrees, the tool approximates non-45 degree edges.

In [Figure 2-9](#), the green lines show the distance between the line and the approximation, which cannot exceed **max_skew_approximation_error**.

Figure 2-9. Stairstep Approximation (VSB11)



- `conversion_address_unit {0.00125 | unit}`

An optional argument specifying the unit size for the Nuflare output in microns. The default value for VSB11 is 0.00125.

- `frame_width {1024 | width}`

An optional argument defining the width of each frame (the y-dimension size of a frame, which is the upper limit of the frame width), specified in microns at mask scale. The default is 512.

This value is an upper limit on the frame width. The actual frame width varies based on the values of `frame_max_data` and `frame_bde_max_data`.

- `cell_max_height {1024 | mheight}`

An optional argument defining the maximum height of any VSB11 cell, specified in microns at mask scale. The default is 1024.

- `cell_max_width {128 | mwidth}`

An optional argument defining the maximum width of any VSB11 cell, specified in microns at mask scale. The default is 128.

- `cell_subfield_size {64 | size}`

An optional argument defining the size of the subfield cell, specified in microns at mask scale. The default is 64 for VSB11.

- `cell_subfield_margin {0.25 | size}`

An optional keyword used to define the subfield margin division, specified in microns at mask scale. The default is 0.25 for VSB11.

- `frame_max_data {256 | fdata}`

An optional argument defining the maximum size of the `frame.X` files, specified in MBs. The value of this argument must not exceed 8191 MB. The default is 256 MB.

- `common_max_data {256 | cdata}`

An optional argument defining the maximum size of the `common.chp` file, specified in MBs. The value of this argument must not exceed 8191 MB. The default is 256 MB.

- `frame_bde_max_data {256 | fbdedata}`

An optional argument defining the maximum BDE size of each of the `frame.X` files, specified in MBs. The value of this argument must not exceed 8191 MB. The default is 256 MB.

- `common_bde_max_data {256 | cbdedata}`

An optional argument defining the maximum BDE size of the `common.chp` file, specified in MBs. The value of this argument must not exceed 8191 MB. The default is 256 MB.

- `runtime_exception_severity bde_filesize_limit [0 | 1]`

An optional argument defining whether Calibre issues an error or warning under certain circumstances. The `bde_filesize_limit` argument defines the behavior of Calibre when it encounters a frame that cannot be split and does not satisfy BDE or file size constraints.

0 — The Calibre FRACTURE operation attempts to finish operations on the remaining frames and executes the remaining statements in the rule file, even though the resulting VSB11 chip is not valid. This is the default setting. The transcript include the following warning: “Warning: MDP exit status: 30”.

1 — The Calibre FRACTURE operation exits when encountering this situation.

- `runtime_exception_severity frame_width_range [0 | 1]`

An optional argument that determines the behavior of VSB11 fracturing operations if the `frame_width` argument is outside the range of 128 to 1024 microns.

0 — The Calibre FRACTURE operation attempts to finish operations on the remaining frames and executes the remaining statements in the rule file, even though the resulting VSB11 chip is not valid. This is the default setting The transcript includes the following warning: “Warning: MDP exit status: 31”.

1 — The Calibre FRACTURE operation exits when encountering this situation.

- `pattern_array_compression {0 | 1}`

An optional argument that enables activation of pattern array compression. The default is 1 (on).

- `pattern_compression {0 | 1}`

An optional argument that enables activation of pattern compression. The default is 1 (on).

- `pattern_set description`

An optional argument specifying the pattern set. The default is 1.

- `chip_name name`

An optional argument specifying the string to output to the `chip.cnf` file as a designator for the chip being fractured, such as “sample-chip”.

name — an alpha-numeric string. By default, the final (right most) portion of the `chip_directory` path is used.

- `make_chip_ini`

If this optional keyword is present, then a `chip.ini` file is created in the same directory level as that of VSB11 chip. By default, Calibre FRACTURE does not create a `chip.ini` file.

- `array_cell_cost_multiplier {4 | multiplier}`

An optional argument controlling how the fracturing operation makes the trade off between fractured file size and the number of cell locations and cell types.

multiplier — defines a cost for creating additional cell types. A large multiplier results in fewer cell types and cell counts but a larger data file and BDE. The default is 4.

Tip

 This value must be kept at default. However, if you run into high remote LVHEAP numbers such that the remotes are at or nearing swap, decrease this value by 2. If further improvements in remote LVHEAP numbers are necessary, you can further decrease the value of this keyword, down to 1.

- `exposed_area`

This keyword directs the program to compute and print the area of the pattern file that is covered by trapezoids. The covered area is printed as a percentage of the fracture region area and in square microns. The information is printed to the Calibre transcript.

Note

 This keyword is only supported in section-based mode.

VSB12 V1 Format

The VSB12 V1-specific format for FRACTURE NUFLARE has its own set of keywords. There are two versions of the VSB12 format. The V1 format is the original VSB12 format. The V2 format uses a more streamlined subset of the original VSB12 keyword set.

Usage

Calibre FRACTURE Syntax

```
FRACTURE NUFLARE layer [...layerN]
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]
[FILENAME output_file_name]
{FILE {parameter_block_name | ['parameter_list']} }
```

where *parameter_list* can contain the following arguments:

Global Calibre FRACTURE Arguments

```
[computation_mode {hier | section}]
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[shift x y]
[magnify mag]
[reverse_tone]
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]
[trapezoid_pass_through]
[trapezoid_groups {map_name annotation}...]
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}}...}]
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]]
[vboasis_precision_multiplier {n/d | AUTO}]
```

```
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]
[-preserve 0 | 1]]
[section_count_limit limit]
```

Common Arguments for Vector Beam Formats

```
[small_value value]
[cd {{internal {cd_max_width cd_min_length | auto}} |
{external cd_min_width cd_max_width cd_min_length}}]
[shot_size s]
[reduce_shot_count {1 | 2}]
[reduce_run_time {0 | 1 | 2}]
```

VSB12 V1-Specific Arguments

version 12

chip_directory *output_directory*

max_skew_approximation_error *skew_error*

```
[block_width {1024 | value}]
[block_height {128 | value}]
[frame_width {512 | width}]
[subframe_width {8 | value}]
[cell_max_height {128 | mheight}]
[cell_max_width {128 | mwidth}]
[frame_max_num_cell_locations {4000000 | num}]
[chip_name name]
[make_chip_ini]
[frame_max_data {256 | fdata}]
[common_max_data {256 | cdata}]
[conversion_address_unit {0.001 | unit}]
[cell_subfield_size {128 | size}]
[cell_subfield_margin {0.5 | size}]
[pattern_composite_rep {0 | 1}]
[pattern_array_compression {0 | 1}]
[pattern_set description]
[pattern_compression {0 | 1}]
[runtime_exception_severity frame_width_range [0 | 1]]
[array_cell_cost_multiplier {4 | multiplier}]
[remote_direct_merge {0 | 1}]
[device {ebm6000 | ebm7000 | ebm8000 | ebm9000}]
[exposed_area]
```

Arguments

- **Global and Common Arguments**

The VSB12 V1 format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23
- “[Global Calibre FRACTURE Arguments](#)” on page 29
- “[Common Arguments for Vector Beam Formats](#)” on page 40

Caution

 The V1 format for VSB12 will be deprecated in future releases, replaced by the V2 format.

- **version 12**

A required argument specifying VSB12 (V1).

- **chip_directory *output_directory***

A required argument specifying the directory path in which to store the fractured data.

For VSB12, the directory is similar in structure to:

```
<output_directory>/  
    chip.cnf  
    cell.common
```

and one of the following:

```
    ref.<n>  
    link.<n>  
    cell.<n>
```

Calibre overwrites the specified directory at the time of execution if it already exists. Environment variables can be used in the path specification of the *output_directory*.

This keyword is mutually exclusive from the FILENAME and **file_name** keywords; Calibre generates an error if you specify both for any one Calibre FRACTURE operation.

Tip

 Use this argument to specify the directory of the fracture output if you specify all the other arguments inline to the Calibre FRACTURE operation.

- **max_skew_approximation_error *skew_error***

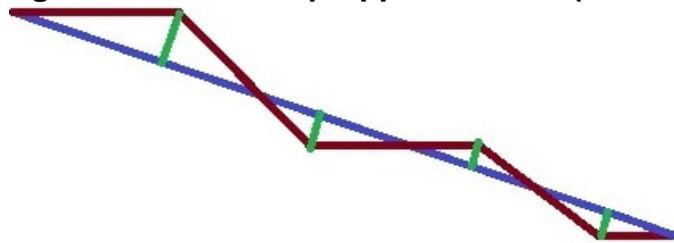
A required argument defining the maximum allowed distance between an approximation of a non-45 degree edge and the original edge.

skew_error — specified in microns at mask scale. The value must be greater than or equal to 1 conversion address unit.

The tool approximates these edges with a staircase created from either horizontal and vertical lines (when close to 90 degrees) or horizontal, vertical, and 45-degree lines (when close to 45 degrees).

In [Figure 2-10](#), the green lines show the distance between the line and the approximation, which cannot exceed **max_skew_approximation_error**.

Figure 2-10. Stairstep Approximation (VSB12)



- **block_width {1024 | value}**

Specifies, in units of microns at output scale, the x-dimension size of a block. The default value is 1024.

- **block_height {128 | value}**

Specifies, in units of microns at output scale, the y-dimension size of a block (the smallest size of a frame). The default value is 128.

- **frame_width {512 | width}**

An optional argument defining the width of each frame (the y-dimension size of a frame, which is the upper limit of the frame width), specified in microns at mask scale. The default is 512.

- **subframe_width {8 | value}**

Specifies, in units of block_width at output scale, the x-dimension size of a subframe. The default value is 8. Note that its value in microns depends on the value of block_width. For instance, at the default setting, the value of subframe_width is 8, and block_width is 1024 microns, so the micron value of subframe_width is actually 8192 microns (1024 x 8).

Tip

If scalability is an issue, decrease this value from the default (8) down to 4. This creates twice as many sections for FRACTURE, thus improving granularity. Changing the value of this keyword is not visible in the output fracture format.

- **cell_max_height {128 | mheight}**

An optional argument defining the maximum height of any VSB12 cell, specified in microns at mask scale. The default is 128.

- **cell_max_width {128 | mwidth}**

An optional argument defining the maximum width of any VSB12 cell, specified in microns at mask scale. The default is 128.

- **frame_max_num_cell_locations** {4000000 | *num*}

An optional argument that limits the number of cell locations in any one frame. The default value is 4000000. In the case where frame splitting is allowed, if the number of cell locations exceeds this number, the frames are split, which is similar to the specification of a smaller `frame_width`. A very small number may not be feasible for a given `block_height`. If frames already have a width of 128, and the number of cell locations exceeds the number specified, then cells are not generated that will lead to a larger file size. Siemens EDA recommends leaving this number at the highest number allowed, its default.

- **chip_name** *name*

An optional argument specifying the string to output to `chip.cnf` file as a designator for the chip being fractured, such as “sample-chip”.

name — Specifies an alpha-numeric string. By default, the final (right most) portion of the **chip_directory** path is used.

- **make_chip_ini**

If this optional keyword is present, then a `chip.ini` file is created in the same directory level as that of VSB12 chip. By default, Calibre FRACTURE does not create a `chip.ini` file.

- **frame_max_data** {256 | *fdata*}

An optional argument, specified in MBs, specifying the maximum value of the per frame summation of file sizes for `cell.X`, `ref.X`, and `link.X`, where *X* is an integer greater than 0 referring to the frame number. The value of this argument must not exceed 8191MB. The default is 256 MB. The default is not recommended. The value must be set as high as allowed by the mask manufacturer.

- **common_max_data** {256 | *cdata*}

An optional argument defining the maximum size of the `cell.common` file, specified in MBs. The value of this argument must not exceed 8191 MB. The default is 256 MB.

- **conversion_address_unit** {0.001 | *unit*}

An optional argument specifying the unit size for the VSB12 output. The VSB12 default is 0.001, which specifies the unit size as 1 nm. You can also specify this value as “1 nm”.

- **cell_subfield_size** {128 | *size*}

An optional argument defining the size of the subfield cell, specified in microns at mask scale. The *size* value must not exceed either `cell_max_height` or `cell_max_width`. The default is 128 for VSB12.

- **cell_subfield_margin** {0.5 | *size*}

An optional keyword used to define the subfield margin division, specified in microns at mask scale. The default is 0.5 for VSB12.

- **pattern_composite_rep** {0 | *1*}

When set to 0, this option deactivates composite pattern representation and enables it when set to 1. The setting reduces file size so setting to 1 is recommended, but the user must take

care that the NuFlare hardware is able to accept data generated with this setting. Not all NuFlare mask writers are able to process data generated with this setting. Refer to the VSB12 standard for further details. The default is 0.

- **pattern_array_compression {0 | 1}**

An optional argument that enables pattern array compression. The default is 1 (on).

- **pattern_set *description***

An optional argument specifying the pattern set. The default is 1.

- **pattern_compression {0 | 1}**

An optional argument that enables pattern compression. The default is 1 (on).

- **runtime_exception_severity frame_width_range [0 | 1]**

An optional argument that determines the behavior of VSB12 fracturing operation if the frame_width argument is outside the range of 128 to 1024 microns.

0 — (default) The Calibre FRACTURE operation attempts to finish operations on the remaining frames and executes the remaining statements in the rule file, even though the resulting VSB12 chip is not valid. The transcript includes the following warning: “WARNING: MDP exit status: 31”.

1 — The Calibre FRACTURE operation exits the Calibre run when encountering this situation.

- **array_cell_cost_multiplier {4 | *multiplier*}**

An optional argument controlling how the fracturing operation makes the trade off between fractured file size and the number of cell locations and cell types.

multiplier — Defines a cost for creating additional cell types. A large multiplier results in fewer cell types and cell counts but a larger data file. The default is 4.

Tip

 This value must be kept at default. However, if you run into high remote LVHEAP numbers such that the remotes are at or nearing swap, decrease this value by 2. If further improvements in remote LVHEAP numbers are necessary, you can further decrease the value of this keyword, down to 1.

- **remote_direct_merge {0 | 1}**

This keyword helps with the scalability of a Calibre MTFlex run with a large amount of disk I/O. Setting this option to 1 means that **chip_directory** is directly accessible by remote nodes; 0 means otherwise (which is the same as not specifying this keyword at all). Because remote processes directly accesses the file system, the **chip_directory** path must be valid on the remote computer. This typically means providing an absolute network path.

- device {ebm6000 | ebm7000 | ebm 8000 | ebm 9000}

This keyword sets default values of certain arguments for different Nuflare E-beam writers. Currently, the default value of keyword shot_size is set according to the value of this keyword:

- ebm6000: default value of shot_size is set to 0.8
- ebm7000: default value of shot_size is set to 0.5
- ebm8000: default value of shot_size is set to 0.35
- ebm9000: default value of shot_size is set to 0.25

If the device keyword is not specified, shot_size is the default, 2.0. If the value of shot_size is specified manually, then that value overrides the default set by the device keyword.

- exposed_area

This keyword directs the program to compute and print the area of the pattern file that is covered by trapezoids. The covered area is printed as a percentage of the fracture region area and in square microns. The information is printed to the Calibre transcript.

Note

 This keyword is only supported in section-based mode.

VSB12i Format

The VSB12i-specific format for FRACTURE NUFLARE has its own specific keywords. VSB12i is similar to the VBS12 format, but also supports compression and allows attribute information assignment to the trapezoids for dose-based mask process corrections.

Usage

Calibre FRACTURE Syntax

```
FRACTURE NUFLARE layer [...layerN]  
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]  
[FILENAME output_file_name]  
{FILE {parameter_block_name | '['parameter_list '']}}
```

where *parameter_list* can contain the following arguments:

Global Calibre FRACTURE Arguments

```
[computation_mode {hier | section}]  
[input_region_severity {0 | 1}]  
[index_options [-topcell cellname]]  
[lint]  
[log_file_name name [-dump]]  
[mirror {x | y | x y | y x}]  
[rotate {0 | 90 | 180 | 270}]  
[shift x y]  
[magnify mag]  
[reverse_tone]  
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]  
[trapezoid_pass_through]  
[<SVRFSTART>  
  svrf_statements  
<SVRFEND>]  
[svrf_range r]  
[embedded_svrf_scale {WAFER | MASK}]  
[svrf_layer_name {layer_name | {layer_name oasis_layer_number  
  [oasis_layer_datatype]}}...}]  
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]]  
[vboasis_precision_multiplier {n/d | AUTO}]
```

```
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]
[-preserve 0 | 1]]
[section_count_limit limit]
```

Common Arguments for Vector Beam Formats

```
[small_value value]
[cd {{internal {cd_max_width cd_min_length | auto}} |
{external cd_min_width cd_max_width cd_min_length}}]
[shot_size s]
[reduce_shot_count {1 | 2}]
[reduce_run_time {0 | 1}]
```

VSB12i-Specific Arguments

```
version 12i
chip_directory output_directory
max_skew_approximation_error skew_error
[block_width {1024 | value}]
[block_height {128 | value}]
[frame_width {512 | width}]
[subframe_width {8 | value}]
[cell_max_height {128 | mheight}]
[cell_max_width {128 | mwidth}]
[frame_max_num_cell_locations {4000000 | num}]
[frame_max_data {256 | fdata}]
[conversion_address_unit {0.001 | unit}]
[cell_subfield_margin {0.5 | size}]
[runtime_exception_severity frame_width_range [0 | 1]]
[device {ebm6000 | ebm7000 | ebm8000 | ebm9000}]
[make_chip_ini]
[exposed_area]
[compression {0 | 1}]
[trapezoid_groups {{map_name annotation}... | -doseMap {map_name dose}...}]
[section_width {8 | value}]
[skew_approximation_mode {0 | 1 | 2}]
```

Arguments

- **Global and Common Arguments**

The VSB12i format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23
- “[Global Calibre FRACTURE Arguments](#)” on page 29
- “[Common Arguments for Vector Beam Formats](#)” on page 40

- **version 12i**
A required argument specifying VSB12i.
- **chip_directory *output_directory***
A required argument specifying the directory path in which to store the fractured data.

For VSB12i, the directory is similar in structure to:

```
<output_directory>/  
    chip.cnf  
    cell.common
```

and one of the following:

```
ref.<n>  
link.<n>  
cell.<n>
```

Calibre overwrites the specified directory at the time of execution if it already exists.
Environment variables can be used in the path specification of the *output_directory*.

This keyword is mutually exclusive from the FILENAME and **file_name** keywords; Calibre generates an error if you specify both for any one fracture operation.

Tip

 Use this argument to specify the directory of the fracture output if you specify all the other arguments inline to the fracture operation.

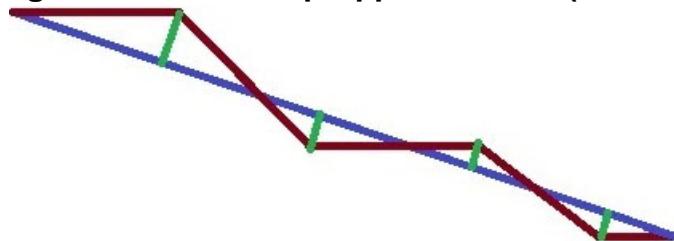
- **max_skew_approximation_error *skew_error***

A required argument, specified at mask scale, defining the maximum allowed distance between an approximation of a non-45 degree edge and the original edge. The *skew_error* value must be greater than or equal to 1 conversion address unit.

The tool approximates these edges with a staircase created from either horizontal and vertical lines (when close to 90 degrees) or horizontal, vertical, and 45-degree lines (when close to 45 degrees).

In [Figure 2-11](#), the green lines show the distance between the line and the approximation, which cannot exceed **max_skew_approximation_error**.

Figure 2-11. Stairstep Approximation (VSB12i)



- `block_width {1024 | value}`

Specifies, in units of microns at output scale, the x-dimension size of a block. The default value is 1024.

- `block_height {128 | value}`

Specifies, in units of microns at output scale, the y-dimension size of a block (the smallest size of a frame). The default value is 128.

- `frame_width {512 | width}`

An optional argument defining the width of each frame (the y-dimension size of a frame, which is the upper limit of the frame width), specified in microns at mask scale. The default is 512.

As the frame splitting feature is not supported in VSB12i, you must set the `frame_width` option to 128 to ensure a legal VSB12i output file.

- `subframe_width {8 | value}`

Specifies, in units of `block_width` at output scale, the x-dimension size of a subframe. The default value is 8. Note that its value in microns depends on the value of `block_width`. For instance, at the default setting, the value of `subframe_width` is 8, and `block_width` is 1024 microns, so the micron value of `subframe_width` is actually 8192 microns (1024 x 8).

Tip

 If scalability is an issue, you must decrease this value from the default (8) down to 4. This creates twice as many sections for the fracture operation, thus improving granularity. Changing the value of this keyword is not visible to the user in the output fracture format.

- `cell_max_height {128 | mheight}`

An optional argument defining the maximum height of any VSB12i cell, specified in microns at mask scale. The default is 128.

- `cell_max_width {128 | mwidht}`

An optional argument defining the maximum width of any VSB12i cell, specified in microns at mask scale. The default is 128.

- `frame_max_num_cell_locations {4000000 | num}`

This limits the number of cell locations in any one frame. The default value is 4000000. In the case where frame splitting is allowed, if the number of cell locations exceeds this number, the frames are split, which is similar to the specification of a smaller `frame_width`. A very small number may not be feasible for a given `block_height`. If frames already have a width of 128, and the number of cell locations exceeds the number specified, then cells are not generated that will lead to a larger file size. Siemens EDA recommends leaving this number at the highest number allowed, its default.

- `frame_max_data {256 | fdata}`

An optional argument, specified in MBs, specifying the maximum value of the per frame summation of file sizes for *cell.X*, *ref.X*, and *link.X*, where *X* is an integer greater than 0 referring to the frame number. The value of this argument must not exceed 8191MB. The default is 256 MB. The default is not recommended. The value must be set as high as allowed by the mask manufacturer.

- `conversion_address_unit {0.001 | unit}`

An optional argument specifying the unit size for the VSB12i output. The VSB12i default is 0.001, which specifies the unit size as 1 nm. You can also specify this value as “1nm”.

- `cell_subfield_margin {0.5 | size}`

An optional keyword used to define the subfield margin division, specified in microns at mask scale. The default is 0.5 for VSB12i.

- `runtime_exception_severity frame_width_range [0 | 1]`

An optional argument that determines the behavior of VSB12i fracturing operation if the `frame_width` argument is outside the range of 128 to 1024 microns.

0 — (default) The Calibre FRACTURE operation attempts to finish operations on the remaining frames and executes the remaining statements in the rule file, even though the resulting VSB12i chip is not valid. The transcript includes the following warning: “WARNING: MDP exit status: 31”

1 — The Calibre FRACTURE operation exits the Calibre run when encountering this situation.

- `device {ebm6000 | ebm7000 | ebm8000 | ebm9000}`

This keyword sets default values of certain arguments for different NuFlare e-beam writers. Currently, the default value of keyword `shot_size` is set according to the value of this keyword:

- ebm6000: default value of `shot_size` is set to 0.8
- ebm7000: default value of `shot_size` is set to 0.5
- ebm8000: default value of `shot_size` is set to 0.35
- ebm9000: default value of `shot_size` is set to 0.25

If the device keyword is not specified, then the default value of `shot_size` is used. If the value of `shot_size` is specified manually, then that value overrides the default set by the device keyword.

- `make_chip_ini`

If this optional keyword is present, then a `chip.ini` file is created in the same directory level as that of VSB12i chip. By default, Calibre FRACTURE does not create a `chip.ini` file.

- exposed_area

This keyword directs the program to compute and print the area of the pattern file that is covered by trapezoids. The covered area is printed as a percentage of the fracture region area and in square microns. The information is printed to the Calibre transcript.

Note



This keyword is only supported in section-based mode.

- compression {0 | 1}

When set to 1, this optional argument activates cell pattern data compression in VSB12i. If set to 0, cell pattern data compression is deactivated. The default value is 1. In VSB12i the cell pattern data is written in compressed form using ZLIB compression. The attribute information can be written using the trapezoid_groups argument.

- trapezoid_groups {{map_name annotation}... | -doseMap {map_name dose}...}

An optional argument that enables multilayer trapezoidal groups. This functionality requires the use of embedded SVRF. There must be a one-to-one match of map_names to embedded SVRF output layer names. Each of these layers are processed and converted to trapezoids separately. The integer annotation can be used in various ways by different fracture formats:

- VSB12i — The annotation is transcribed to Attribute Information; all trapezoids in a group have the same Attribute Information (PH4 header) whose value is the annotation for that group.

Specifying -doseMap indicates dose values are used instead of annotations. The dose value must be a number in the range of 0.0 and 2.0. If a value of 1.0 is specified in the list of dose values, the maximum number of dose values that can be specified is eight; otherwise, the maximum number is seven. The annotations are assigned internally as follows:

```
trapezoid_groups -doseMap L1 0.8 L2 0.75 L3 1.0 L4 1 .1
                  L5 1 .0 L6 1.2 L7 1.3     L8 1.4
```

The annotations for this specification are:

L1	0.8	Annotation	1
L2	0.75	Annotation	2
L3	1.0	Annotation	0
L4	1 .1	Annotation	3
L5	1 .0	Annotation	4
L6	1.2	Annotation	5
L7	1.3	Annotation	6
L8	1.4	Annotation	7

If the dose value list does not have a value of 1.0 specified, the annotation values are assigned integer values sequentially from 1 to n , where n is the number of dose values specified. Regardless of whatever value list is specified in trapezoid_groups, the standard dose value of 1.0 is always assumed to be present with its annotation set to 0.

- `section_width {8 | value}`

Specified in units of `block_width` at output scale. This is the x-dimension size of a section. It has a default value of 8. Note its value in microns depends on the value of `block_width`. For instance, at default, the value of `section_width` is 8 and that of `block_width` is 1024 microns, so the value of `section_width` is 8192 microns. This argument is reset to `subframe_width` if found to be greater than `subframe_width`.

- `skew_approximation_mode {0 | 1 | 2}`

An optional argument that sets which algorithm is used to approximate skew edges.

- 0 — Generates trapezoids with 45 degree edges. This is the default setting.
- 1 — Generates trapezoids with Manhattan edges. This improves the approximation in terms of runtime and data quality.
- 2 — Generates both 45 degree and Manhattan edges (similar to mode 0) but also heuristically determines horizontal or vertical 1D trapezoids in the skew edge regions.

Description

There are a number of notes that apply for FRACTURE NUFLARE.

- Specify the unit size in terms of the `conversion_address_unit`, where you can specify 0.001(for VSB12 and VSB12i) or 0.00125 (for VSB11).
- The FRACTURE operation outputs the frames into the directory you specify with the `chip_directory` keyword.
- VSB11 output consists of three types of files:
 - **Frame Files** — One file for each horizontal stripe of trapezoid data and cell placements. Each file is named using the convention “frame.X”, where X is greater than 0.
 - **Common File** — The trapezoid data that can be referenced from multiple frame files. The filename is `common.chp`.
 - **Chip Configuration File** — The conversion arguments that define the data conversions that must be performed as part of the drawing process. The filename is `chip.cnf`.

The file names are predefined. You control the name of the directory containing these files through the FILENAME keyword to the Fracture SVRF statement or the `chip_directory` Inline keyword. If this directory exists, and contains files, the files are overwritten. Additionally, the Calibre FRACTUREt operation supports VSB11 large common cells during fracture operations.

- For VSB12i, the output chip directory is given the same name as the job deck directory name.

- The VSB12 and VSB12i fracture format is a vector-beam format and, like the VSB11 fracture format, stores the fractured data in multiple files (three files per frame instead of one) inside a single directory. For VSB12 and VSB12i, trapezoid data and cell placements are divided into horizontal stripes called frames. Information for each frame is stored in a separate set of files (one of *cell.X*, *ref.X*, and *link.X*, where *X* is an integer greater than 0 referring to the frame number).
- The origin of the output data is always located at the lower left corner.
- The frame splitting feature is not supported in VSB12 (V2) and VSB12i. This means that you must set the frame_width option to 128 um to ensure a legal VSB12i output file.
- For VSB12i formats, the job deck is created automatically with a dose map, but it is not created for explicit dose values.

VSB12 V2 Format

The VSB12 V2 format for FRACTURE NUFLARE has its own specific keywords. There are two versions of the VSB12 format. The V1 format is the original VSB12 format. The V2 format uses a more streamlined subset of the original VSB12 keyword set. Note that the frame-splitting feature is not supported for V2, and you must instead specify the minimum frame width (128 um) to ensure a legal VSB12 output file.

Usage

Calibre FRACTURE Syntax

```
FRACTURE NUFLARE layer [...layerN]  
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]  
[FILENAME output_file_name]  
{FILE {parameter_block_name | '['parameter_list '']} }
```

where *parameter_list* can contain the following arguments:

Global Calibre FRACTURE Arguments

```
[computation_mode {hier | section}]  
[input_region_severity {0 | 1}]  
[index_options [-topcell cellname]]  
[lint]  
[log_file_name name [-dump]]  
[mirror {x | y | x y | y x}]  
[rotate {0 | 90 | 180 | 270}]  
[shift x y]  
[magnify mag]  
[reverse_tone]  
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]  
[trapezoid_pass_through]  
[trapezoid_groups {map_name annotation}...]  
<SVRFSTART>  
    svrf_statements  
<SVRFEND>]  
[svrf_range r]  
[embedded_svrf_scale {WAFER | MASK}]  
[svrf_layer_name {layer_name | {layer_name oasis_layer_number  
    [oasis_layer_datatype]}...} ]
```

```
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]]
[vboasis_precision_multiplier {n/d | AUTO}]
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]
    [-preserve 0 | 1]]
[section_count_limit limit]
```

[Common Arguments for Vector Beam Formats](#)

```
[small_value value]
[cd {{internal {cd_max_width cd_min_length | auto}} | 
    {external cd_min_width cd_max_width cd_min_length}}]
[shot_size s]
[reduce_shot_count {1 | 2}]
[reduce_run_time {0 | 1 | 2}]
```

VSB12 V2-Specific Arguments

```
version 12_v2
chip_directory output_directory
max_skew_approximation_error skew_error
[block_width {1024 | value}]
[block_height {128 | value}]
[frame_width {512 | width}]
[subframe_width {8 | value}]
[cell_max_height {128 | mheight}]
[cell_max_width {128 | mwidth}]
[frame_max_num_cell_locations {4000000 | num}]
[frame_max_data {256 | fdata}]
[conversion_address_unit {0.001 | unit}]
[cell_subfield_margin {0.5 | size}]
[pattern_composite_rep {0 | 1}]
[runtime_exception_severity frame_width_range [0 | 1]]
[device {ebm6000 | ebm7000 | ebm8000 | ebm9000}]
[make_chip_ini]
[exposed_area]
[shot_count_stats]
[section_width {8 | value}]
[skew_approximation_mode {0 | 1 | 2}]
```

Arguments

- **Global and Common Arguments**

The VSB12 V2 format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23

- “[Global Calibre FRACTURE Arguments](#)” on page 29
- “[Common Arguments for Vector Beam Formats](#)” on page 40
- **version 12_v2**
A required argument specifying VSB12 (V2) format.
- **chip_directory output_directory**
A required argument specifying the directory path in which to store the fractured data.
For VSB12, the directory is similar in structure to:

```
<output_directory>/  
    chip.cnf  
    cell.common
```

and one of the following:

```
ref.<n>  
link.<n>  
cell.<n>
```

Calibre overwrites the specified directory at the time of execution if it already exists.
Environment variables can be used in the path specification of the *output_directory*.

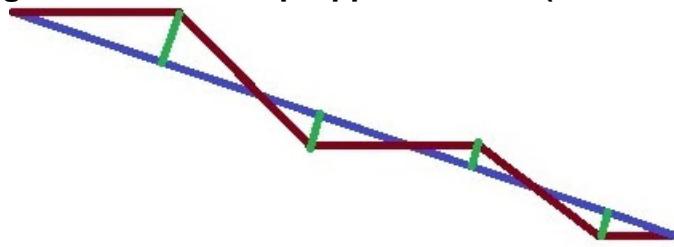
This keyword is mutually exclusive from the FILENAME and **file_name** keywords; Calibre generates an error if you specify both for any one fracture operation.

Tip

 Use this argument to specify the directory of the fracture output if you specify all the other arguments inline to the fracture operation.

- **max_skew_approximation_error skew_error**
A required argument, specified at mask scale, defining the maximum allowed distance between an approximation of a non-45 degree edge and the original edge. The *skew_error* value must be greater than or equal to 1 conversion address unit.
The tool approximates these edges with a staircase created from either horizontal and vertical lines (when close to 90 degrees) or horizontal, vertical, and 45-degree lines (when close to 45 degrees).
In [Figure 2-12](#), the green lines show the distance between the line and the approximation, which cannot exceed **max_skew_approximation_error**.

Figure 2-12. Stairstep Approximation (VSB12_V2)



- `block_width {1024 | value}`

Specifies, in units of microns at output scale, the x-dimension size of a block. The default value is 1024.

- `block_height {128 | value}`

Specifies, in units of microns at output scale, the y-dimension size of a block (the smallest size of a frame). The default value is 128.

- `frame_width {512 | width}`

An optional argument defining the width of each frame (the y-dimension size of a frame, which is the upper limit of the frame width), specified in microns at mask scale. The default is 512.

As the frame splitting feature is not supported in VSB12 (V2), you must set the `frame_width` option to 128 to ensure a legal VSB12 (V2) output file.

- `subframe_width {8 | value}`

Specifies, in units of `block_width` at output scale, the x-dimension size of a subframe. The default value is 8. Note that its value in microns depends on the value of `block_width`. For instance, at the default setting, the value of `subframe_width` is 8, and `block_width` is 1024 microns, so the micron value of `subframe_width` is actually 8192 microns (1024 x 8).

Tip

If scalability is an issue, you must decrease this value from the default (8) down to 4. This creates twice as many sections for the fracture operation, thus improving granularity. Changing the value of this keyword is not visible to the user in the output fracture format.

- `cell_max_height {128 | mheight}`

An optional argument defining the maximum height of any VSB12 cell, specified in microns at mask scale. The default is 128.

- `cell_max_width {128 | mwidht}`

An optional argument defining the maximum width of any VSB12 cell, specified in microns at mask scale. The default is 128.

- `frame_max_num_cell_locations {4000000 | num}`

This optional argument limits the number of cell locations in any one frame. The default value is 4000000. In the case where frame splitting is allowed, if the number of cell locations exceeds this number, the frames are split, which is similar to the specification of a smaller `frame_width`. A very small number may not be feasible for a given `block_height`. If frames already have a width of 128, and the number of cell locations exceeds the number specified, then cells are not generated that will lead to a larger file size. Siemens EDA recommends leaving this number at the highest number allowed, its default.

- `frame_max_data {256 | fdata}`

An optional argument, specified in MBs, specifying the maximum value of the per frame summation of file sizes for `cell.X`, `ref.X`, and `link.X`, where `X` is an integer greater than 0 referring to the frame number. The value of this argument must not exceed 8191MB. The default is 256 MB. The default is not recommended. The value must be set as high as allowed by the mask manufacturer.

- `conversion_address_unit {0.001 | unit}`

An optional argument specifying the unit size for the VSB12 output. The VSB12 default is 0.001, which specifies the unit size as 1 nm. You can also specify this value as “1 nm”.

- `cell_subfield_margin {0.5 | size}`

An optional argument used to define the subfield margin division, specified in microns at mask scale. The default is 0.5 for VSB12.

- `pattern_composite_rep {0 | 1}`

When set to 0, this optional argument deactivates composite pattern representation and enables it when set to 1. The setting reduces file size so setting to 1 is recommended, but the user must take care that the NuFlare hardware is able to accept data generated with this setting. Not all NuFlare mask writers are able to process data generated with this setting. Refer to the VSB12 standard for further details. The default is 0.

- `runtime_exception_severity frame_width_range [0 | 1]`

An optional argument that determines the behavior of VSB12 fracturing operation if the `frame_width` argument is outside the range of 128 to 1024 microns.

0 — (default) The Calibre FRACTURE operation attempts to finish operations on the remaining frames and executes the remaining statements in the rule file, even though the resulting VSB12 chip is not valid. The transcript includes the following warning: “WARNING: MDP exit status: 31”.

1 — The Calibre FRACTURE operation exits the Calibre run when encountering this situation.

- device {ebm6000 | ebm7000 | ebm8000 | ebm9000}

This keyword sets default values in microns of certain arguments for different NuFlare e-beam writers. Currently, the default value of keyword shot_size is set according to the value of this keyword:

- ebm6000: default value of shot_size is set to 0.8
- ebm7000: default value of shot_size is set to 0.5
- ebm8000: default value of shot_size is set to 0.35
- ebm9000: default value of shot_size is set to 0.25

If the device keyword is not specified, then the default value of shot_size is used. If the value of shot_size is specified manually, then that value overrides the default set by the device keyword.

- make_chip_ini

If this optional keyword is present, then a chip.ini file is created in the same directory level as that of VSB12 chip. By default, Calibre FRACTURE does not create a chip.ini file.

- exposed_area

If this keyword is present then the transmission density is output. Transmission density is the percentage of the transmission area relative to the extent of the input layer together with the absolute transmission coverage area. It is expressed in um².

Note



This keyword is only supported in section-based mode.

- shot_count_stats

If this keyword is present, then figure and shot count statistics are output to the fracture transcript. This keyword performs the same functions as the Calibre MDPview vsbTrapStats utility (refer to the *Calibre MDPview User's and Reference Manual* for information) and uses the following arguments for the calculation:

- *small_figure_size* — A floating point number that specifies, in microns, a minimum size for figures. The *small_figure_size* value is set to be equal the user-defined fracture *small_value* value.
- *shot_size* — A floating point number, in microns. The default value corresponds to the different mask writers. The value for *shot_size* is equal to the user-defined fracture *shot_size* value.

Note that you will not need to enter values for the *shot_size* and *small_figure_size*.

- section_width {8 | *value*}

Specified in units of block_width at output scale. This is the x-dimension size of a section. It has a default value of 8. Note its value in microns depends on the value of block_width. For instance, at default, the value of section_width is 8 and that of block_width is 1024 microns,

so the value of section_width is 8192 microns. This argument is reset to subframe_width if found to be greater than subframe_width.

- `skew_approximation_mode {0 | 1 | 2}`

An optional argument that sets which algorithm is used to approximate skew edges.

- 0 — Generates trapezoids with 45 degree edges. This is the default setting.edges.
- 1 — Generates trapezoids with Manhattan edges. This improves the approximation in terms of runtime and data quality.
- 2 — Generates both 45 degree and Manhattan edges (similar to mode 0) but also heuristically determines horizontal or vertical 1D trapezoids in the skew edge regions.

FRACTURE_NUFLARE_MBF (FRACTUREn)

SVRF mask data preparation command for NUFLARE_MBF formats. NUFLARE_MBF is a fracture format for Nuflare multibeam machines. The format is similar to VSB12i with additional constructs.

Usage

Calibre FRACTURE Syntax

```
FRACTURE NUFLARE_MBF layer [...layerN]
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]
[FILENAME output_file_name]
{FILE {parameter_block_name | '['parameter_list '']} }
```

where *parameter_list* can contain the following arguments:

Global Calibre FRACTURE Arguments

```
[computation_mode {hier | section}]
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[shift x y]
[magnify mag]
[reverse_tone]
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]
[trapezoid_groups {map_name annotation}...]
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}}...]
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]]
[vboasis_precision_multiplier {n/d | AUTO}]
```

```
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]
[-preserve 0 | 1]]
[section_count_limit limit]
```

MBF-Specific Arguments

version 1.0

```
chip_directory output_directory
subframe_max_data {256 | value}
grid_subframe_division {on | off}
auto_grid_subframe_division {on | off}
[block_width {1024 | value}]
[block_height {82 | value}]
[conversion_address_unit {0.001 | unit}]
[frame_width {82 | width}]
[subframe_width {8 | value}]
[section_width {8 | value}]
[cell_max_height {frame width val | mheight}]
[cell_max_width {frame width val | mwidth}]
[frame_max_num_cell_locations {4000000 | num}]
[frame_max_data {256 | fdata}]
[cell_subfield_margin {0.5 | size}]
[runtime_exception_severity frame_width_range [0 | 1]]
[pattern_composite_rep {0 | 1}]
[device mbm1000]
[make_chip_ini]
[compression {0 | 1}]
```

Arguments

- **Global and Common Arguments**

The VSB12 V2 format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23
- “[Global Calibre FRACTURE Arguments](#)” on page 29
- “[Common Arguments for Vector Beam Formats](#)” on page 40

- **version 1.0**

A required argument specifying NUFLARE_MBF version 1.0 format to be generated.

- **chip_directory *output_directory***

A required argument specifying the directory path in which to store the fractured data.

For NUFLARE_MBF, the directory is similar in structure to:

```
<output_directory>/  
    chip.cnf  
    cell.common
```

and one of the following:

```
    ref.<n>  
    link.<n>  
    cell.<n>
```

Calibre overwrites the specified directory at the time of execution if it already exists.
Environment variables can be used in the path specification of the *output_directory*.

This keyword is mutually exclusive from the FILENAME and **file_name** keywords; Calibre generates an error if you specify both for any one fracture operation.

Tip

 Use this argument to specify the directory of the fracture output if you specify all the other arguments inline to the fracture operation.

- **subframe_max_data {256 | value}**

A required argument that specifies the maximum value of the per subframe summation of file sizes for files cell.X, link.X and ref.X specified in units of MB (megaBytes), where X is a frame number. In this case, per subframe summation is the summation of the localized cell, ref, and link data for that subframe. The value of this argument should not exceed 8191MB. The default value is 256.

- **grid_subframe_division {on | off}**

A required argument that, when set to on, enables grid subframe division. The chip is divided in the form of a grid by frames and subframes using the frame_width and subframe_width keywords. If the auto_grid_subframe_division keyword is enabled along with this argument, then subframes that overflow are further divided within this grid. If automatic grid is not enabled, subframes may overflow. A warning is then issued in the transcript. The default value is off.

- **auto_grid_subframe_division {on | off}**

A required argument that, when set to on, enables automatic grid subframe division. If this keyword is enabled along with grid_subframe_division, the chip is divided in the form of a grid by frames and subframes using the frame_width and subframe_width keywords. Subframes that overflow are further divided within this grid. If this argument is enabled without grid subframe division, the chip is divided in such a way that the resulting frames and subframes satisfy the general restrictions of frames and subframes as described in NUFLARE_MBF format specification. The default value is off.

- **block_width {1024 | value}**

Specifies, in units of microns at output scale, the x-dimension size of a block. The default value is 1024.

- `block_height {82 | value}`

An optional argument that specifies, in units of microns at output scale, the y-dimension size of a block (the smallest size of a frame). The default value is 82.

- `conversion_address_unit {0.001 | unit}`

An optional argument specifying the unit size for the NUFLARE_MBF output. The NUFLARE_MBF default is 0.001, which specifies the unit size as 1 nm. You can also specify this value as “1nm”.

- `frame_width {82 | width}`

An optional argument defining the width of each frame (the y-dimension size of a frame, which is the upper limit of the frame width), specified in microns at mask scale. The default is 82.

As the frame splitting feature is not supported in NUFLARE_MBF, you must set the `frame_width` option to 128 to ensure a legal NUFLARE_MBF output file.

- `subframe_width {8 | value}`

An optional argument that specifies, in units of `block_width` at output scale, the x-dimension size of a subframe. The default value is 8. Note that its value in microns depends on the value of `block_width`. For instance, at the default setting, the value of `subframe_width` is 8, and `block_width` is 1024 microns, so the micron value of `subframe_width` is actually 8192 microns (1024 x 8).

Tip

 If scalability is an issue, you must decrease this value from the default (8) down to 4.

This creates twice as many sections for the fracture operation, thus improving granularity. Changing the value of this keyword is not visible to the user in the output fracture format.

- `section_width {8 | value}`

An optional argument that is specified in units of `block_width` at output scale. This is the x-dimension size of a section. It has a default value of 8. Note its value in microns depends on the value of `block_width`. For instance, by default, the value of `section_width` is 8 and that of `block_width` is 1024 microns, so the value of `section_width` is 8192 microns. This argument is reset to `subframe_width` if found to be greater than `subframe_width`.

- `cell_max_height {frame_width_val | mheight}`

An optional argument defining the maximum height of any VSB12i cell, specified in microns at mask scale. The default is the same as the `frame_width` value.

- `cell_max_width {frame_width_val | mwidth}`

An optional argument defining the maximum width of any VSB12i cell, specified in microns at mask scale. The default is the same as the `frame_width` value.

- `frame_max_num_cell_locations {4000000 | num}`

An optional argument that limits the number of cell locations in any one frame. The default value is 4000000. In the case where frame splitting is allowed, if the number of cell locations exceeds this number, the frames are split, which is similar to the specification of a smaller `frame_width`. A very small number may not be feasible for a given `block_height`. If frames already have a width of 128, and the number of cell locations exceeds the number specified, then cells are not generated that will lead to a larger file size. Siemens EDA recommends leaving this number at the highest number allowed, its default.

- `cell_subfield_margin {0.5 | size}`

An optional argument used to define the subfield margin division, specified in microns at mask scale. The default is 0.5 for NUFLARE_MBF.

- `frame_max_data {256 | fdata}`

An optional argument, specified in MBs, specifying the maximum value of the per frame summation of file sizes for `cell.X`, `ref.X`, and `link.X`, where `X` is an integer greater than 0 referring to the frame number. The value of this argument must not exceed 8191MB. The default is 256 MB. The default is not recommended. The value must be set as high as allowed by the mask manufacturer.

- `runtime_exception_severity frame_width_range [0 | 1]`

An optional argument that determines the behavior of VSB12i fracturing operation if the `frame_width` argument is outside the range of 128 to 1024 microns.

`0` — (default) The Calibre FRACTURE operation attempts to finish operations on the remaining frames and executes the remaining statements in the rule file, even though the resulting VSB12i chip is not valid. The transcript includes the following warning: “WARNING: MDP exit status: 31”

`1` — The Calibre FRACTURE operation exits the Calibre run when encountering this situation.

- `pattern_composite_rep {0 | 1}`

When set to 0, this optional argument deactivates composite pattern representation and enables it when set to 1. The setting reduces file size so setting to 1 is recommended, but the user must take care that the NuFlare hardware is able to accept data generated with this setting. Not all NuFlare mask writers are able to process data generated with this setting. Refer to the NUFLARE_MBF standard for further details. The default is 0.

- `device mbm1000`

An optional argument that sets default values of certain arguments for different NUFLARE_MBF writers. Currently, only mbm1000 is supported. The default values for mbm1000 are as follows:

- `frame_width: 82 um`
- `block_width: 1024um`
- `block_height: 82um`

- `cell_max_width` and `cell_max_height` are set to `frame_width`
- `make_chip_ini`
If this optional keyword is present, then a *chip.ini* file is created in the same directory level as that of NUFLARE_MBF chip. By default, Calibre FRACTURE does not create a *chip.ini* file.

Description

There are a number of notes that apply for FRACTURE NUFLARE_MBF.

- Specify the unit size in terms of the *conversion_address_unit*, where you can specify 0.001 for MBF.
- The FRACTURE operation outputs the frames into the directory you specify with the **chip_directory** keyword.
- The output consists of three types of files:
 - **Frame Files** — One file for each horizontal stripe of trapezoid data and cell placements. Each file is named using the convention “frame.*X*”, where *X* is greater than 0.
 - **Common File** — The trapezoid data that can be referenced from multiple frame files. The filename is *common.chp*.
 - **Chip Configuration File** — The conversion arguments that define the data conversions that must be performed as part of the drawing process. The filename is *chip.cnf*.

The file names are predefined. You control the name of the directory containing these files through the FILENAME keyword to the Fracture SVRF statement or the **chip_directory** Inline keyword. If this directory exists, and contains files, the files are overwritten.

- The MBF fracture format is a vector-beam format and stores the fractured data in multiple files (three files per frame instead of one) inside a single directory. For MBF, trapezoid data and cell placements are divided into horizontal stripes called frames. Information for each frame is stored in a separate set of files (one of *cell.X*, *ref.X*, and *link.X*, where *X* is an integer greater than 0 referring to the frame number).
- The origin of the output data is always located at the lower left corner.
- The frame splitting feature is not supported for NUFLARE_MBF. This means that you must set the `frame_width` option to 128 um to ensure a legal NUFLARE_MBF output file.

FRACTURE OASIS_MAPPER (FRACTUREp)

SVRF mask data preparation command for OASIS.MAPPER formats. OASIS.MAPPER is the proprietary file format for MAPPER Lithography.

Note

 In FRACTURE OASIS_MAPPER, geometries are always shifted so that the lower left of the total extent or the user-specified extent is (0, 0).

Usage

Calibre FRACTURE Syntax

```
FRACTURE OASIS_MAPPER layer [...layerN]  
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]  
[FILENAME output_file_name]  
{FILE {parameter_block_name | '['parameter_list '']}}
```

where *parameter_list* can contain the following arguments:

Global Calibre FRACTURE Arguments

```
file_name output_file_name  
[computation_mode {hier | section}]  
[input_region_severity {0 | 1}]  
[index_options [-topcell cellname]]  
[lint]  
[log_file_name name [-dump]]  
[mirror {x | y | x y | y x}]  
[rotate {0 | 90 | 180 | 270}]  
[magnify mag]  
[reverse_tone]  
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]  
[trapezoid_pass_through]  
[trapezoid_groups {map_name annotation}...]  
<SVRFSTART>  
    svrf_statements  
<SVRFEND>]  
[svrf_range r]  
[embedded_svrf_scale {WAFER | MASK}]
```

```
[svrf_layer_name {layer_name | {layer_name oasis_layer_number  
[oasis_layer_datatype]}...}]  
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]]  
[vboasis_precision_multiplier {n[/d] | AUTO}]  
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]  
[-preserve 0 | 1]]
```

OASIS_MAPPER-Specific Arguments

fracture_units gridsize

computation_mode section

[cell_size_x cx cell_size_y cy]

[stitching_strategy [soft_edge_all | smart_boundary | auto_soft_edge]]

[num_dose_levels n]

[section_density cells_in_x_dir cells_in_y_dir]

Arguments

- **fracture_units gridsize**

A required argument specifying the number of output units per micron for the OASIS output file.

gridsize — an integer that satisfies the equation:

$$\text{gridsize} = 1000/x$$

where *x* is the desired grid size in microns, for example:

Desired Grid Size	gridsize
2 nm	500
4 nm	250

Note



Because *gridsize* must be an integer, the desired unit size (in nanometers) must evenly divide 1000.

- **computation_mode section**

A required argument specifying a section-based fracture method during processing that improves scalability and performance when processing flat designs. If you use Embedded SVRF (the <SVRFSTART> and <SVRFEND> block), then you must use section mode only.

- **cell_size_x cx cell_size_y cy**

An optional argument that specifies the maximum cell size. Both keywords must be used together. For OASIS_MAPPER, the *cx* value is 2.2 um. The supported range for *cy* is [100 -

1000] um. If `cell_size_x` or `cell_size_y` are not specified, the default values are used. The default for `cell_size_x` is 2.2 um and the default for `cell_size_y` is 1000 um.

- `stitching_strategy` [`soft_edge_all` | `smart_boundary` | `auto_soft_edge`]]

An optional argument that specifies the stitching strategy to be applied in the overlap region between stripes.

`soft_edge_all` — Consists of a gradual fade-out of the geometry's dose in the overlap region towards the border of a stripe, allowing the next stripe to supercede by adding the complement to obtain the right dose on the wafer. The maximum number of dose levels can be set using the `num_dose_levels` keyword.

`smart_boundary` — Ensures that a critical portion of a feature is written by beam only, provided the feature is not longer than the width of the overlap region of 200 nm. For features longer than 200 nm, soft edges are applied. The maximum number of dose levels can be set using the `num_dose_levels` keyword. Siemens EDA recommends this option and it is the default setting.

`auto_soft_edge` — Ensures that features in the stitching area are present twice, once in a cell belonging to the left stripe, and once in a cell belonging to the right stripe. The rasterizer then applies a continuous gradient soft edge throughout all stitching regions.

- `num_dose_levels n`

An optional argument that specifies the maximum number of dose levels allowed in the mapper output, including 100% dose. For example, if `num_dose_levels` is set to 5, the overlap region between stripes is split into four different regions with 80%, 60%, 40% and 20% relative dose (as the percentage of maximum dose of 100%). The default value is 5 if the `smart_boundary` and `soft_edge_all` are options for the `stitching_strategy` keyword, and 1 if `auto_soft_edge` option is selected.

- `section_density cells_in_x_dir cells_in_y_dir`

This argument allows the user to control the amount of data handled by a section. `cells_in_x_dir` and `cells_in_y_dir` together specify the section size. If not specified, default values are used.

`cells_in_y_dir` — A positive integer that specifies the number of level-2 cells per section in the x direction. The allowed range is [1, 13000].

`cells_in_y_dir` — A positive integer that specifies the number of level-2 cells per section in the y direction. The allowed range is [1, `max_cells_in_y_direction`], where `max_cells_in_y_direction = upper_bound(oasis_mapper_field_dimension_in_y / cell_size_y)`. Smaller values give larger section counts, which can distribute better on large clusters and help keep remote RAM usage low.

The default value is 125 for `cells_in_x_dir` and 1 for `cells_in_y_dir`.

Description

Calibre Fracture generates an OASIS.MAPPER format that conforms to v03-05 of the OASIS.MAPPER format specification. OASIS cell localization and all properties associated with it are supported. Files generated by FRACTURE OASIS_MAPPER can be used wherever

OASIS and VB:OASIS files can be used. Refer to the OASIS.MAPPER specification for a full description of the format.

FRACTURE OASIS_MAPPER requires Calibre RDS (Remote Data Server) to reduce local (server) host memory requirements and to improve Calibre MTFlex performance by distributing data across remote hosts. Calibre RDS can be activated by enabling -remotedata keyword on command line for Calibre MTFlex invocation. Using Calibre without -remotedata (for single threaded or multi threaded modes) may increase memory usage and run time for larger data. Refer to the [Calibre Administrator's Guide](#) for more information about Calibre RDS.

FRACTURE OASIS_MAPPER adheres to the OASIS.MAPPER requirements that the spec requires output file size should not exceed 1.5TB and the stripe volumes do not exceed 2GB. These violations are enforced during the fracture process and any error messages are written to the transcript. FRACTURE OASIS_MAPPER also enforces a limit of 13000 stripes and a maximum stripe length of 33 mm. Calibre verifies the post-fractured data extent limit to be within 26000.2 um x 33000 um and any error messages are written to the transcript.

FRACTURE OASIS_MASK (FRACTUREv)

SVRF mask data preparation command for the OASIS.MASK format.

Usage

Calibre FRACTURE Syntax

```
FRACTURE OASIS_MASK layer [...layerN]  
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]  
[FILENAME output_file_name]  
{FILE {parameter_block_name | '['parameter_list '']}}
```

where *parameter_list* can contain the following arguments:

Global Calibre FRACTURE Arguments

```
file_name output_file_name  
[computation_mode {hier | section}]  
[input_region_severity {0 | 1}]  
[index_options [-topcell cellname]]  
[lint]  
[log_file_name name [-dump]]  
[mirror {x | y | x y | y x}]  
[rotate {0 | 90 | 180 | 270}]  
[magnify mag]  
[reverse_tone]  
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]  
[trapezoid_pass_through]  
<SVRFSTART>  
    svrf_statements  
<SVRFEND>  
[svrf_range r]  
[embedded_svrf_scale {WAFER | MASK}]  
[svrf_layer_name {layer_name | {layer_name oasis_layer_number  
    [oasis_layer_datatype]}}...]}]  
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]]  
[vboasis_precision_multiplier {n/d | AUTO}]  
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]  
    [-preserve 0 | 1]]
```

[section_count_limit *limit*]

Common Arguments for Vector Beam Formats

[small_value *value*]

[cd {{internal {cd_max_width cd_min_length | auto}} |
{external cd_min_width cd_max_width cd_min_length}}]

[shot_size *s*]

[reduce_shot_count {1 | 2}]

[reduce_run_time {0 | 1 | 2}]

OASIS_MASK-Specific Arguments

fracture_units *gridsize*

[cell_size *cs* | cell_size_x *cx* cell_size_y *cy*]

[depth *d*]

[oasis_vsb_conformance_severity {0 | 1 | 2}]

[cblock]

[exposed_area]

[positive_output_extent]

[speed_mode]

trapezoid_groups {{map_name annotation}... | -doseMap {map_name dose}...}

Arguments

- **Global and Common Arguments**

The FRACTURE OASIS_MASK format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23
- “[Global Calibre FRACTURE Arguments](#)” on page 29
- “[Common Arguments for Vector Beam Formats](#)” on page 40

- **fracture_units** *gridsize*

A required argument specifying the number of output units per micron for the OASIS output file.

gridsize — an integer that satisfies the equation:

$$\textit{gridsize} = 1000/x$$

where *x* is the desired grid size in microns, for example:

Desired Grid Size	<i>gridsize</i>
2 nm	500
4 nm	250

Note

 Because **gridsize** must be an integer, the desired unit size (in nanometers) must evenly divide 1000.

- **cell_size** *cs*

Specifies the maximum allowed cell size in the output in um. The default size is 1000 um.

- **cell_size_x** *cx* **cell_size_y** *cy*

An alternative method of specifying the maximum cell size. Both keywords must be used together. This specification method is mutually exclusive with **cell_size**. The default size for both is 1000 um.

Note

 A combination of small cell size and larger fracture extent may cause the number of fracture sections to exceed the file descriptor limit. If this situation is found in the rule file, the compiler issues an error message and provides an estimate of the larger cell size setting to avoid the conflict. The larger cell size is an estimate that depends on cell size, fracture extent, and soft-boundary values.

- **depth** *d*

Specifies the maximum allowed hierarchy depth in the output. The hierarchy tree depth is optimally reduced to allow maximum compression. A depth of zero levels is the top cell only; a setting of one level indicates top cell plus geographic tile cells; a setting of two levels means top cell, tile cells and flat cover cells; and so on. The default is 1, the recommended value for optimal file size.

- **oasis_vsb_conformance_severity** {0 | 1 | 2}

Specifies the conformance strictness to OASIS.MASK format. 0 indicates that the output OASIS.MASK file strictly conforms. 1 and 2 indicate no conformance is required; however when value 1 is specified, a warning message is output. When the option is set to 0, this implies the following:

- The file PRECISION value must evenly divide 10000 or equal 800.
- Cells other than the top cell are constrained to have Cartesian dimensions in the range [10, 1000] um.
- Hierarchy depth must be one or two levels. The depth argument must be 1 or 2.

- **cblock**

When **cblock** is specified, the output OASIS file writes CBLOCK records as specified in the OASIS format to reduce the size of the file.

- **exposed_area**

This keyword directs the program to compute and print the area of the pattern file that is covered by trapezoids. The covered area is printed as a percentage of the fracture region area and in square microns. The information is printed to the Calibre transcript.

- **positive_output_extent**

If this keyword is present, a shift is applied to fracture data as a last part of the transform to ensure that the transformed fracture extent lower left corner is placed at the origin of the output coordinate system.

- **speed_mode**

If present, all mask writer objectives are ignored in an attempt to produce correct OASIS.MASK output as quickly as possible. This data is not suitable for direct input to mask-writer tools, because the data is composed of one-dimensional trapezoids with no effort to avoid split CDs or small trapezoids. It is, however, useful as a standard hand-off format of trapezoided data and useful for purposes of geometrical processing. Using “speed_mode” processing is significantly faster than the default OASIS.MASK trapezoiding.

This keyword requires section mode computation and is incompatible with the trapezoid_group option.

- **trapezoid_groups { {map_name annotation}... | -doseMap {map_name dose}... }**

An optional keyword that enables multilayer trapezoidal groups. This functionality requires the use of embedded SVRF. There must be a one-to-one match of map_names to embedded SVRF output layer names. Each of these layers are processed and converted to trapezoids separately. The integer annotation can be used in various ways by different fracture formats:

- OASIS * — The annotation is transcribed to the datatype, where all trapezoids in a group have a datatype whose value is the annotation for that group.

Specifying -doseMap indicates dose values are used instead of annotations. The dose value must be a number in the range of 0.0 and 2.0. If a value of 1.0 is specified in the list of dose values, the maximum number of dose values that can be specified is eight; otherwise, the maximum number is seven. The annotations are assigned internally as follows:

```
trapezoid_groups -doseMap L1 0.8 L2 0.75 L3 1.0 L4 1 .1
                  L5 1 .0 L6 1.2 L7 1.3     L8 1.4
```

The annotations for this specification are:

L1	0.8	Annotation	1
L2	0.75	Annotation	2
L3	1.0	Annotation	0
L4	1.1	Annotation	3
L5	1.0	Annotation	4
L6	1.2	Annotation	5
L7	1.3	Annotation	6
L8	1.4	Annotation	7

If the dose value list does not have a value of 1.0 specified, the annotation values are assigned integer values sequentially from 1 to n , where n is the number of dose values specified. Regardless of whatever value list is specified in trapezoid_groups, the standard dose value of 1.0 is always assumed to be present with its annotation set to 0.

Description

There are a number of notes that apply to FRACTURE_OASIS_MASK.

- The unit size as specified in terms of the fracture units, which is expressed as (1/desired grid size in microns).
- For an OASIS.MASK pattern file, there are no restrictions on the filename.
- For an OASIS.MASK pattern file, the origin is where it is in the input.
- For FRACTURE_OASIS_MASK, in the log file, you get a total trapezoid count similar to the following:

```
TOTAL TRAPEZOID COUNT. U: 0 L: 3427284562
```

This line in the transcript represents the total trapezoid count represented using only 32 bit numbers; the actual total trapezoid count is $U * 2^{32} + L$. For this example, the total trapezoid count is $0 * 2^{32} + 3427284562 = 3427284562$. U is zero only if the trapezoid count is less than 2^{32} (4,294,967,296).

FRACTURE OASIS_MBW (FRACTUREi)

SVRF mask data preparation command for OASIS.MBW formats. OASIS.MBW is a mask writer format for IMS machines compatible with OASIS.MASK format. All utilities that accept OASIS.MASK also work with OASIS.MBW.

Usage

Calibre FRACTURE Syntax

```
FRACTURE OASIS_MBW layer [...layerN]  
[INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]  
[FILENAME output_file_name]  
{FILE {parameter_block_name | '['parameter_list '']} }
```

where ***parameter_list*** can contain the following arguments:

Global Calibre FRACTURE Arguments

```
file_name output_file_name  
[computation_mode {hier | section}]  
[input_region_severity {0 | 1}]  
[index_options [-topcell cellname]]  
[lint]  
[log_file_name name [-dump]]  
[mirror {x | y | x y | y x}]  
[rotate {0 | 90 | 180 | 270}]  
[magnify mag]  
[reverse_tone]  
[direct_FS_access [INPUT] [INTERMEDIATE] [OUTPUT]]  
[trapezoid_pass_through]  
<SVRFSTART>  
    svrf_statements  
<SVRFEND>]  
[svrf_range r]  
[embedded_svrf_scale {WAFER | MASK}]  
[svrf_layer_name {layer_name | {layer_name oasis_layer_number  
                  [iosis_layer_datatype]}...} ]  
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]]  
[vboasis_precision_multiplier {n/d | AUTO}]
```

```
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-withIndex]
[-preserve 0 | 1]]
[section_count_limit limit]
```

OASIS_MBW-Specific Arguments

fracture_units *gridsize*

version {1.0 | 1.1 | 1.2 | 2.0 | 2.1}

computation_mode section

[merged_direction [NOT_MERGED | X_MERGED | Y_MERGED]]

[cell_size *cs* | cell_size_x *cx* cell_size_y *cy*]

[section_density *cells_in_x* *cells_in_y*]

[monotone {NONE | X | Y | XY}]

[cblock]

[exposed_area]

[positive_output_extent]

[trapezoid_groups {{*map_name annotation*}... | -doseMap {*map_name dose*}...}]

Arguments

- **Global and Common Arguments**

The FRACTURE OASIS_MBW format has its own specific arguments. For the other common arguments, refer to the following sections:

- “[Calibre FRACTURE Syntax](#)” on page 23
- “[Global Calibre FRACTURE Arguments](#)” on page 29

- **fracture_units *gridsize***

A required argument specifying the number of output units per micron for the OASIS output file.

gridsize — an integer that satisfies the equation:

$$\textit{gridsize} = 1000/x$$

where *x* is the desired grid size in microns, for example:

Desired Grid Size	<i>gridsize</i>
2 nm	500
4 nm	250

Note

 Because *gridsize* must be an integer, the desired unit size (in nanometers) must evenly divide 1000.

- **version {1.0 | 1.1 | 1.2 | 2.0 | 2.1}**

A required argument that specifies the data format version for the output file.

1.0 — The MBW_OVERLAP_FREE property is added, set to a value of 1.

1.1 — The default version.

1.2 — The CELL_COVERAGE property is added.

2.0 — POLYGON records are supported.

2.1 — Monotone POLYGONS are supported.

- **computation_mode** section

A required argument that specifies a section-based fracture method during processing that improves scalability and performance when processing flat designs. If you use Embedded SVRF (the <SVRFSTART> and <SVRFEND> block), then you must use section mode only.

- **merged_direction** [NOT_MERGED | X_MERGED | Y_MERGED]

Specifies the direction of a merge operation.

NOT_MERGED — Do not merge.

X_MERGED — Merge in the x-direction.

Y_MERGED — Merge in the y-direction.

- **cell_size** *cs*

An optional argument that specifies the maximum allowed cell size in the output in um. The range is 10 to 100 um for version 1.0 and 10 to 50 um for versions above 1.0. The default size is 50 um.

- **cell_size_x** *cx* **cell_size_y** *cy*

An alternative optional method of specifying the maximum cell size. Both keywords must be used together. This specification method is mutually exclusive with **cell_size**. The default size for both is 1000 um.

Note

 A combination of small cell size and larger fracture extent may cause the number of fracture sections to exceed the file descriptor limit. If this situation is found in the rule file, the compiler issues an error message and provides an estimate of the larger cell size setting to avoid the conflict. The larger cell size is an estimate that depends on cell size, fracture extent, and soft-boundary values.

- **section_density** *cells_in_x* *cells_in_y*

An optional argument that specifies the number of cells to be included in a section defined in x- and y-dimensions. The maximum value of *cells_in_x* is limited by the number of columns and *cells_in_y* is limited by the number of rows. The default value set is (10 10).

- **monotone** [NONE | X | Y | XY]

This argument, required only if version 2.1 is specified, sets the behavior for generation of monotone polygons.

NONE — Generates non-monotone polygons (similar to version 2.0)

X — Generates only x-monotone polygons.

Y — Generates only y-monotone polygons.

XY — Generates x- and y-monotone polygons.

- **cblock**

An optional keyword that specifies that the output OASIS file writes CBLOCK records as specified in the OASIS format to reduce the size of the file.

- **exposed_area**

This optional keyword directs the program to compute and print the area of the pattern file that is covered by trapezoids. The covered area is printed as a percentage of the fracture region area and in square microns. The information is printed to the Calibre transcript.

- **positive_output_extent**

If this optional keyword is present, a shift is applied to fracture data as a last part of the transform to ensure that the transformed fracture extent lower left corner is placed at the origin of the output coordinate system.

- **trapezoid_groups { {map_name annotation}... | -doseMap {map_name dose}... }**

An optional argument that enables multilayer trapezoidal groups. This functionality requires the use of embedded SVRF. There must be a one-to-one match of map_names to embedded SVRF output layer names. Each of these layers are processed and converted to trapezoids separately. The integer annotation can be used in various ways by different fracture formats:

- OASIS * — The annotation is transcribed to the datatype, where all trapezoids in a group have a datatype whose value is the annotation for that group.

Specifying -doseMap indicates dose values are used instead of annotations. The dose value must be a number in the range of 0.0 and 2.0. If a value of 1.0 is specified in the list of dose values, the maximum number of dose values that can be specified is eight; otherwise, the maximum number is seven. The annotations are assigned internally as follows:

```
trapezoid_groups -doseMap L1 0.8 L2 0.75 L3 1.0 L4 1 .1
                  L5 1 .0 L6 1.2 L7 1.3      L8 1.4
```

The annotations for this specification are:

L1	0.8	Annotation	1
L2	0.75	Annotation	2
L3	1.0	Annotation	0
L4	1.1	Annotation	3
L5	1.0	Annotation	4
L6	1.2	Annotation	5
L7	1.3	Annotation	6
L8	1.4	Annotation	7

If the dose value list does not have a value of 1.0 specified, the annotation values are assigned integer values sequentially from 1 to n , where n is the number of dose values

specified. Regardless of whatever value list is specified in trapezoid_groups, the standard dose value of 1.0 is always assumed to be present with its annotation set to 0.

Description

The FRACTURE OASIS_MBW process produces three types of data in conformance with OASIS.MBW file format:

- X-merged — Polygons are fractured to maximize fractured trapezoid dimensions in X direction.
- Y-merged — Polygons are fractured to maximize fractured trapezoid dimensions in Y direction.
- Not-merged — No steps are taken to maximize fractured trapezoid dimension.

All keywords valid for OASIS.MASK fracture are also valid for this format except for the following:

- depth — This keyword is not valid as the hierarchy depth of OASIS.MBW format is fixed to 1.
- speed_mode
- small_value — This keyword is not valid as OASIS.MBW is a multibeam format and VSB shots are not involved.
- cd — This keyword is not valid as OASIS.MBW is a multibeam format and VSB shots are not involved.
- shot_size — This keyword is not valid as OASIS.MBW is a multibeam format and VSB shots are not involved.
- oasis_vsb_conformance_severity

There are a number of notes that apply to FRACTURE OASIS_MBW.

- The unit size as specified in terms of the fracture units, which is expressed as (1/desired grid size in microns).
- For an OASIS.MBW pattern file, there are no restrictions on the filename.
- For an OASIS.MBW pattern file, the origin is where it is in the input.
- For FRACTURE OASIS_MBW, in the log file, you get a total trapezoid count similar to the following:

```
TOTAL TRAPEZOID COUNT. U: 0 L: 3427284562
```

This line in the transcript represents the total trapezoid count represented using only 32 bit numbers; the actual total trapezoid count is $U * 2^{32} + L$. For this example, the total

trapezoid count is $0*2^{32}+3427284562 = 3427284562$. U is zero only if the trapezoid count is less than 2^{32} (4,294,967,296).

Considerations for Calibre FullScale Runs

Calibre MDP can be run on the Calibre® FullScale™ platform. However, there are some limitations.

Calibre FullScale Overview

Calibre FullScale is a separately licensed Calibre engine that improves scalability and runtime for typical post-tapeout designs. Invoke Calibre FullScale with “calibre -pto” instead of “calibre -drc -hier.”

Edge results may differ between -pto and -drc -hier runs due to skew edges crossing section boundaries. The skew edges can be from either the input or the intermediate layers. For example, operations such as External REGION can generate non-Manhattan edges.

Calibre FullScale does not support all SVRF operations². If a -pto run is started with a rule file that contains unsupported operations, the run exits with an error entitled “Forbidden operations:” followed by the unsupported operations. Generally, Boolean and topological DRC operations are supported, as well as a subset of the DFM, Litho, RET, and MDP keywords. See “[Calibre FullScale SVRF Support](#)” in the *Calibre Post-Tapeout Flow User’s Manual* for a complete list.

Calibre FullScale cannot be used for LVS, PERC, or parasitic extraction. It also cannot be used with hyperscaling (-hyper).

Changes to Transcript

The Calibre FullScale platform produces a different transcript than the Calibre hierarchical engine. The transcripts also include all the information traditionally reported by the Calibre hierarchical engine. For more details on interpreting the transcript, see “[Interpreting the Calibre Transcript for PTO Applications](#)” in the *Calibre Post-Tapeout Flow User’s Manual*.

Hardware Requirements

The number of remote hosts can stress the primary host. The memory load can be distributed across the cluster by using remote data servers (RDS). When you use a cluster, you must invoke Calibre FullScale with -remotedata. 8 GB per core is recommended. See “[Calibre FullScale Platform](#)” in the *Calibre Post-Tapeout Flow User’s Manual* for more information on RDS.

RDS benefits from high bandwidth. To avoid saturating the network, the bandwidth should be greater than 1 Gbit per sec.

2. Most SVRF specifications are supported. Operations are functions which, when evaluated, create a derived layer.

Calibre MDP in Calibre FullScale

Calibre MDP rule files may require some changes to run when run on the Calibre FullScale platform. Check the rule file for restrictions concerning these FRACTURE operations and keywords in FullScale mode:

- INSIDE OF EXTENT and INSIDE OF LAYER are not supported.
- Embedded SVRF is not supported.
- The vboasis_path keyword is not supported. Instead, direct input is performed through the LAYOUT PATH SVRF command.
- Currently, only JEOL and VSB12 formats are supported.

Chapter 3

Section-Based Processing, Calibre Embedded SVRF, and MDP EMBED

Calibre® MDP Embedded SVRF™ provides a method of performing SVRF operations for section-based processing of design databases.

Section-Based Processing	117
Embed SVRF Commands in FRACTURE or MDPVERIFY Statements	119
Embed SVRF Commands in an MDP EMBED Statement	123
MDP EMBED.....	126
SVRF Command Support in Embedded SVRF	138
Influence the Interaction Range (BOUND Keyword)	143
Converting a Rule File to Embedded SVRF	145
Embedded SVRF With Multiple Outputs in Calibre MDPverify and MDP EMBED ..	147
MDP EMBED Output to HDB and DDO	148
Processor Counts for MDP EMBED in the Transcript	150

Section-Based Processing

Section-based processing is a method of data processing that partitions design databases into uniformly sized “sections” of data to be processed independently of each other. This method of processing is primarily intended for large databases that have very little or no hierarchical cell efficiency. Because of this, section-based processing is typically used for post-OPC data operations such as Fracture and Mask Rule Checking (MRC).

In contrast to hierarchical data processing that executes dedicated tasks on a cell-by-cell basis, section-based processing executes a sequence of tasks on each section of partitioned data. This process can provide improved performance and scalability in instances where the inherent cell structure in the design database cannot be leveraged efficiently by hierarchical processing.

For each section:

1. Data in the section, and possibly in the buffer region around the section, is requested from the hierarchical database (HDB) or directly from the disk for Direct Data Entry (DDE) applications.
2. The embedded SVRF operations are executed. Any generated data outside the section is removed and the results are generated.

3. The output is reassembled. Note that the outputs are flat.

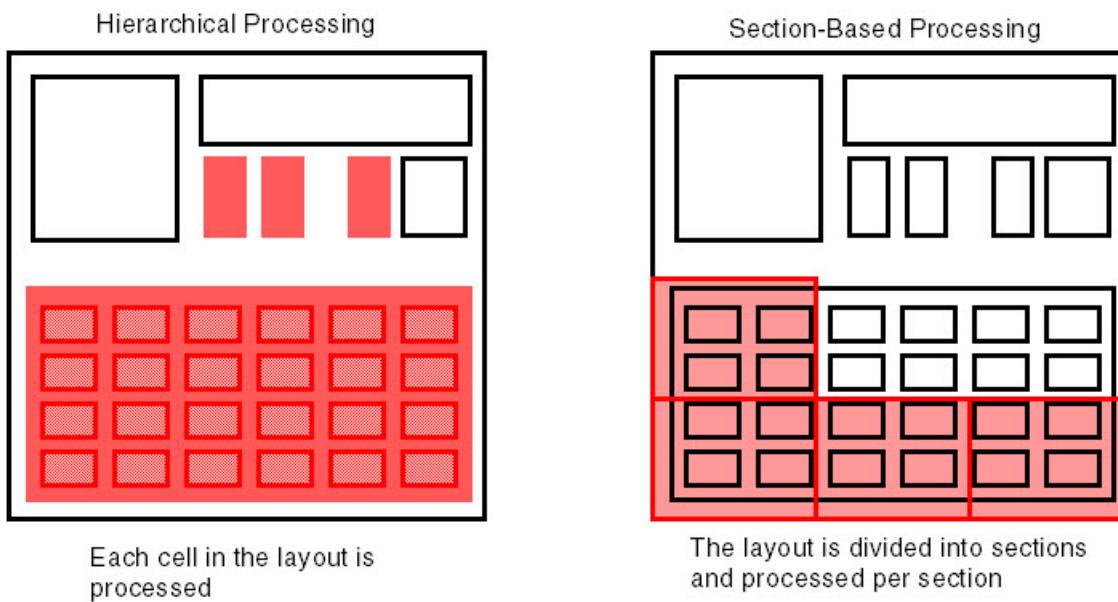
Processing in each section can be done independently on a separate CPU. Section processing attempts to maximize parallel computing potential rather than minimize computation. Section processing ignores potential redundancies when traversing cells by flattening the hierarchy.

In SVRF, you can use section-based processing within the FRACTURE, MDPVERIFY, MDP EMBED, or DENSITY CONVOLVE commands.

Note

- Do not use embedded SVRF in FRACTURE on a heterogeneous cluster (for example, mixed 32-bit and 64-bit clusters), as it can generate geometry errors in the fracture output.
-

Figure 3-1. Hierarchical Versus Section-Based Processing



Embed SVRF Commands in FRACTURE or MDPVERIFY Statements

Calibre MDP gives you the ability to embed a group of SVRF commands within a FRACTURE or MDPVERIFY statement. This allows you to combine other mask data operations efficiently into the same section processing.

Usage

The following is an example for FRACTURE:

```
FRACTURE type layer1 [layerN...]
  [INSIDE OF {EXTENT | x1 y1 x2 y2 | LAYER layer2}]
  [FILENAME output_file_name]
  FILE {[parameter_block_name] | '['
    <SVRFSTART>
    svrf_statements
    <SVRFEND>
    general_and_format_specific_parameters
  ']'}
```

For MDPVERIFY:

```
MDPVERIFY layer1 [...layerN]
  [{INSIDE OF x1
  y1 x2 y2 | INSIDE OF LAYER layer_ext | INSIDE OF EXTENT}]
  [FILENAME name_of_file] [MAP mapstring] FILE {[parameter_block_name] | '['
    verify_type {JEOL2DB | VSB2DB | MEBES2DB | HITACHI2DB |
      JEOL2JEOL | MEBES2MEBES | VSB2VSB | HITACHI2HITACHI |
      MEBES2VSB | MEBES2JEOL | MEBES2HITACHI | MEBES2MICRONIC |
      MICRONIC2DB | MICRONIC2MICRONIC | JEOL2VSB |
      JEOL2HITACHI | VSB2HITACHI | VBOASIS2DB |
      VBOASIS2JEOL | VBOASIS2VSB | VBOASIS2VBOASIS |
      VBOASIS2MEBES | VBOASIS2MICRONIC}
    input_file file1 [file2]
    [<SVRFSTART>
      svrf_statements
    <SVRFEND>]
    MDPVERIFY_parameters
    ']}
  }
```

Arguments

- ‘[’ ... ‘]’

Square brackets must surround the complete block of output definition parameters appearing after the keyword FILE. In addition:

- Output definition parameters must be on lines STRICTLY BETWEEN the left and right brackets (that is, cannot be on the same line).
- Argument text can not contain either a left bracket ([) or right bracket (]). Replace them with <LB> and <RB> respectively.
- You can include comments within the section of the operation set off by square brackets if you begin the comments with a double forward slash (//). These comment characters indicate that all text until the next newline is comment text. Use of the “/* */” style of comments are not permitted.

- <SVRFSTART>

An optional keyword indicating the beginning of the embedded SVRF statement group. This must be on its own line.

- *svrf_statements*

Calibre truncates any SVRF line to 1023 characters if you exceed this value.

You must specify a single rule check within the embedded SVRF statement group. The output of this rule check is what Calibre uses as the input to the FRACTURE operation, and must only be polygon data, not edge or error data. Refer to the *Calibre Verification User’s Manual* for more information about rule checks.

You cannot specify left- or right-brackets within the embedded SVRF statement group. Therefore, any occurrence of brackets must be replaced as follows:

Left bracket ([)	<LB>
Right bracket (])	<RB>

The PRECISION of numeric values in the embedded SVRF statements must match that of the Calibre rule deck containing the FRACTURE statement.

- <SVRFEND>

An optional keyword indicating the end of the embedded SVRF statement group. This must be on its own line.

Note

 The left and right angle brackets (<>) for <SVRFSTART>, <SVRFEND>, <LB>, and <RB> are literal and required. SVRFSTART and SVRFEND must be capitalized.

The given SVRF set is executed effectively on each section before the section is fractured. To avoid errors at section edges, an additional buffer is added to the data before the SVRF is

executed. The buffer is discarded after SVRF processing, but before fracturing. This is the reason that operations with “unbounded” interaction ranges are not allowed. The program calculates the required buffer for the given SVRF operation set. The buffer value calculation can be overridden using the following parameter.

- `svrf_range r`

Specifies the buffer size for section processing. The default is determined automatically and is specified in user units at mask scale.

All SVRF operations occur after the rotation and mirror transform operations. By default, the embedded rule file is applied before magnification and unit conversion. Hence the PRECISION setting of the embedded rule deck matches that of the Calibre rule deck, and all operations work at the same scale as in the Calibre rule deck. Embedded SVRF may optionally be applied at the mask scale instead, in which case the PRECISION setting of the rule file is the fracture file PRECISION setting. The default wafer scale is useful for instance in applying wafer OPC inside the FRACTURE operation. On the other hand, something like mask process correction is applied at the mask scale.

- `embedded_svrf_scale {WAFER| MASK}`

Specifies the scale at which embedded SVRF is applied.

If embedded SVRF is used with the vboasis_path keyword, both HDB and OASIS DDE data may be input to the embedded operation. The SVRF layer name for the OASIS DDE layer must be specified with the following keyword:

- `svrf_layer_name {layer_name | [{layer_name oasis_layer_number [oasis_layer_datatype]}]...}`

A keyword that specifies the names, numbers, and optional data types of the embedded SVRF layers corresponding to OASIS DDE geometry. For a single layer, the first form is used. For multiple layers, each name and corresponding layer number in the OASIS file must be specified. When OASIS DDE input is used with HDB input, the PRECISION settings of the HDB and OASIS file must match.

Note

 Embedded SVRF may produce minor differences in results between HDB and Direct Data Entry runs of 1 DBU or less due to different rounding in merging due to the way each handles sequencing of merging skew region output.

Description

The MDP EMBED command provides section-based processing for SVRF commands.

Note

 Embedded SVRF does not support *job2job* operations (such as MEBESJOB2VSBJOB, JEOLJOB2JEOLJOB, and so on).

Examples

Calibre uses the output of the following embedded SVRF as the input data to the FRACTURE operation. The output must be a derived polygon layer. Calibre issues a parse error if the output is a derived edge or an output layer.

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "sample.oas"
LAYOUT PRIMARY "TOPCELL"
DRC RESULTS DATABASE "out.oas" OASIS PSEUDO

DRC MAXIMUM RESULTS ALL

LAYER poly 0
LAYER fill 10
LAYER marker 20

prime_frac {FRACTURE MEBES poly fill marker FILE [
    mode 5
    address_size 0.02
    file_name SAMPLEPOL.PF
    computation_mode section

    <SVRFSTART>

        sized = SIZE poly BY 0.004 INSIDE OF marker
        all = sized OR fill

        allout {COPY all}

    <SVRFEND>
]
}

}
```

Embed SVRF Commands in an MDP EMBED Statement

The MDP EMBED command provides section-based processing for some SVRF commands, essentially providing section-based DRC capability.

Section-based processing is an alternative which can improve scalability and turnaround time for some limited cases such as mask rule checks (which are DRC-like checks on mask data). These operations are typically, but not always, characterized by two factors:

- The input data is flat or severely degraded hierarchy.
- The checks are local and produce few outputs.

Caution

 No hierarchical efficiency is present in the output layer. It is recommended that any subsequent processing does not use standard hierarchical construction; either use section processing or a binned hierarchical construction method.

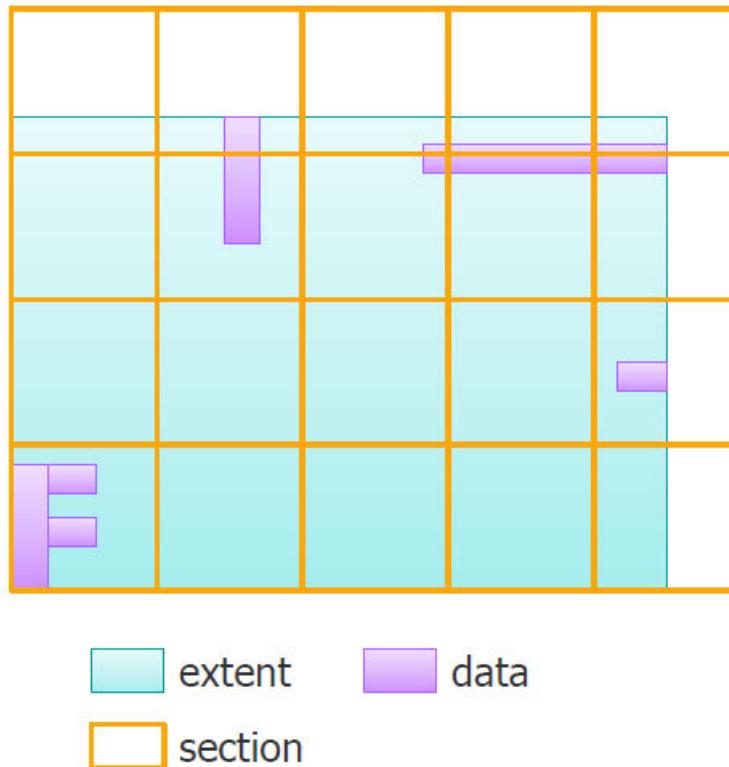
MDP EMBED does not support edge input (this will generate an error).

MDP EMBED enables SVRFs to be processed in section mode. This means that data is partitioned into sections and SVRFs are executed on the data within each section. A small amount of peripheral data are included to ensure correctness near the section's boundary.

MDP EMBED uses a section-centric method of returning data to the user. It returns data for all of the sections overlapping the user-specified hierarchical database (HDB). If you attempt to run on a clip of the data, you cannot control the clip size of the returned data; it will always be a multiple of the section size (unless the section falls off the edge of the data, then the final section is shrink-wrapped to the available OASIS output).

In the following figure, MDP EMBED creates square sections across a specified extent. By default, the section size is 250 um, and the extent is the initial HDB's extent. The section size can be changed using the `section_size` keyword.

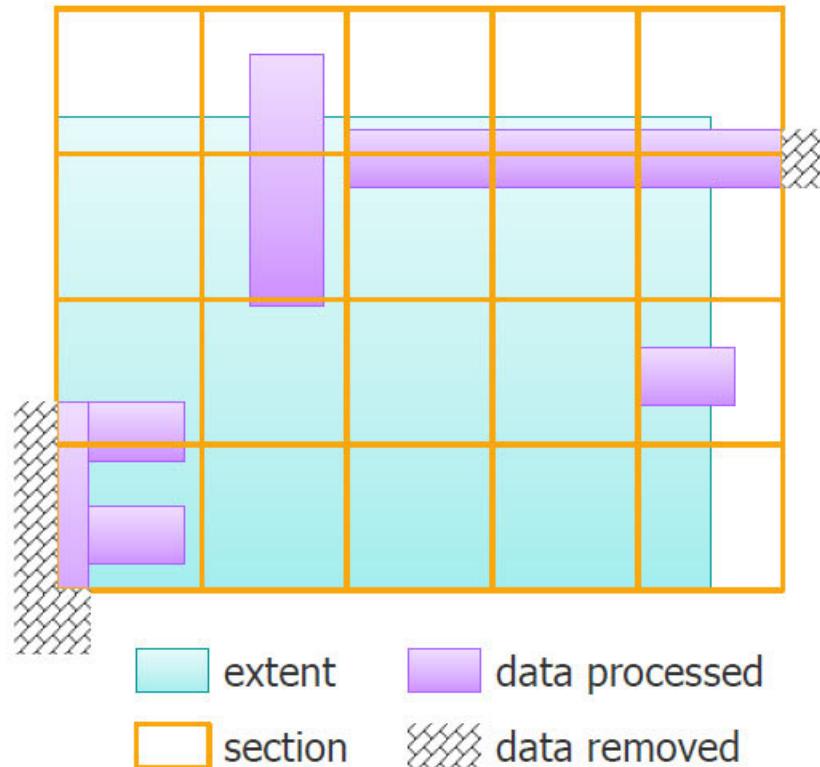
Figure 3-2. MDP EMBED — Sectioning the Data



The section size can be changed using the section_size keyword in MDP EMBED. A different extent can be chosen using the data_extent keyword.

The following figure illustrates that the data within the section's interaction distance are processed by the sequence of SVRFs. The interaction distance is determined by Calibre based on the individual SVRF's interaction range. This behavior can be overridden using the svrf_range keyword in MDP EMBED. Only the result within the section's boundary is kept after processing. The data extending outside the section boundaries are removed.

Figure 3-3. MDP EMBED — Processing the SVRF



Short or limited range SVRFs are supported. In this case, the output is identical to the hierarchical result. Long or unlimited range SVRFs are supported with a bounding range specified. The output accuracy is limited to the specified range value.

MDP EMBED 126

MDP EMBED

SVRF mask data preparation command for section-based processing.

Usage

```
MDP EMBED input_layer [...input_layerN] [MAP output_layer [EDGE]]  
{FILE litho_file_block |  
 FILE[''  
 [section_size sec_value]  
 maximum_output_count count  
 [svrf_range r]  
  
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]]  
[svrf_layer_name {layer_name |  
     {layer_name oasis_layer_number [oasis_layer_datatype]}...}]]  
[vboasis_precision_multiplier {n[/d] | AUTO}]  
[vboasis_layout_magnify n[/d]]  
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1] [-force]]  
  
[vboasis_path2 filepath [-noCheck] [-fastindex [layer datatype]]]  
[svrf_layer_name2 {layer_name |  
     {layer_name oasis_layer_number [oasis_layer_datatype]}...}]]  
[vboasis_precision_multiplier2 {n[/d] | AUTO}]  
[vboasis_layout_magnify2 n[/d]]  
[vboasis_injection2 [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]  
  
[vboasis_path3 filepath [-noCheck] [-fastindex [layer datatype]]]  
[svrf_layer_name3 {layer_name |  
     {layer_name oasis_layer_number [oasis_layer_datatype]}...}]]  
[vboasis_precision_multiplier3 {n[/d] | AUTO}]  
[vboasis_layout_magnify3 n[/d]]  
[vboasis_injection3 [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]  
  
[vboasis_path4 filepath [-noCheck] [-fastindex [layer datatype]]]  
[svrf_layer_name4 {layer_name |  
     {layer_name oasis_layer_number [oasis_layer_datatype]}...}]]  
[vboasis_precision_multiplier4 {n[/d] | AUTO}]  
[vboasis_layout_magnify4 n[/d]]  
[vboasis_injection4 [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]  
  
jobdeck[2] path[-ejobdeck_clipping_severity {1 | 2}] [-ejobdeck_pattern_suffix suffix]  
[-data_search_path path]  
[data_extent {HDB | FILE | FILE2 | FILE3 | FILE4 | DIRECTIN}]  
[index_options [-topcell cellname] [-parallelCblocks]]  
[oasis_output_path {-RDB | [-perSection] path | -INDEX [NOVIEW]}]
```

```
[oasis_output_layers {layer [datatype] | {map_name layer [datatype]}...}]
[oasis_output_primary [HDB | DIRECTIN | FILE | -explicit topcell_name]
[oasis_output_options [-cblock] [-maximum_vertex {mv | ALL}] [-precision val]
    [-magnify val]]
[direct_FS_access [INPUT] [INTERMEDIATE]]
[mrc_only {0 | 1}]
[mdp_remote_tmp dir]
[<SVRFSTART>
    svrf_statements
<SVRFEND>]
'J'
}
```

Description

The MDP EMBED command provides section-based processing for SVRF commands.

Arguments

- ***input_layer*[...*input_layerN*]**

A required argument that specifies the name of an original or derived polygon layer(s). This is the layer that contains the data to which Calibre applies the embedded SVRF functionality.

- **MAP *output_layer* [EDGE]**

An optional keyword that specifies the output layer. When not specified, the one output layer in the embedded SVRF statement group is used for the output of the MDP EMBED operator. This parameter is necessary when there are multiple output layers, in which case there must be exactly one MDP EMBED statement for each output layer. By default, the MAP *output_layer* string is used as the output layer name in the OASIS direct output files (although the -perSection mode does not support layer names). All layers in the output file contain layer names that match the MAP *output_layer* string. If the output is an edge-type layer, then the EDGE keyword must also be used.

- **FILE *litho_file_block***

An optional argument that specifies that the map size parameters are contained in a reusable parameter block defined using the LITHO FILE SVRF statement.

- **FILE '[' ... ']**

A required keyword that specifies that the map size parameters are contained inline within the square brackets ([]). Square brackets must surround all parameters appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:

- Keywords and parameters must be on lines strictly between the left and right brackets (that is, cannot be on the same line).

- You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text. The “/* */” style of comments are not permitted.
- **section_size *sec_value***
An optional argument that specifies the desired size (height and width) of the square section in user units. No implication is made between this parameter and the location of “boundaries” in embedded SVRF. The default is 250 microns.
- **maximum_output_count *count***
A required argument that specifies the limit of number shapes for output from MDP EMBED. The default is to output all shapes (recommended). The default is all.
- **svrf_range *r***
An optional argument that specifies the buffer size for section processing. The default is determined automatically and is specified in user units.
- **vboasis_path[2 | 3 | 4] *filepath* [-noCheck] [-fastindex [*layer datatype*]]**
An optional argument that allows MDP EMBED to take input from an OASIS DDE file. The data is placed on an SVRF layer named by the next keyword. To use this functionality, the Calibre database must be non-MCPO, or a run time error results.
The embedded SVRF is computed within the extent of the Calibre top cell, so the OASIS top cell extent must be explicitly assigned, either through the dummy layer extent called from the HDB or by using the *data_extent* keyword. A run time warning is issued if there is no overlap.
If vboasis_path2, vboasis_path3, and vboasis_path4 are specified (in conjunction with vboasis_path), this means that a second, third, and fourth OASIS input are needed.
The -noCheck option can be specified when checks on the input OASIS DDE layout file are not possible because it has not been generated. This would typically happen when the file is expected to be generated by another SVRF command in the same rule file and thus is not available during parsing for checking purpose. When using -noCheck, use extra care when specifying the file path and associated PRECISION values to avoid errors that are encountered much later.
The -fastIndex keyword enables a “fast mode” of index generation if the input file is an OASIS.MASK file. The file created in fast mode is valid only if the input OASIS.MASK file contains a single layer-datatype pair. Fast mode index generation should be avoided when there are multiple layers or datatypes in the input. By default, fast mode index generation assumes layer-datatype of 0-0. You can override these values by specifying layer-datatype pair as an argument to -fastIndex option.
Several restrictions and caveats apply to the use of this alternate input method, documented in the Direct OASIS Input arguments described in “[Global Calibre FRACTURE Arguments](#)” on page 29.

Compressed layouts (those ending in the extension .gz) are not supported by vboasis_path.

- svrf_layer_name[2 | 3 | 4]{*layer_name* | {*layer_name oasis_layer_number*
[*oasis_layer_datatype*]}}...}

An optional argument that specifies the names, numbers, and optional data types of the embedded SVRF layers corresponding to OASIS DDE geometry. If embedded SVRF is used with the vboasis_path keyword, both HDB and OASIS DDE data may be input to the embedded operation. The SVRF layer names for the OASIS DDE layers must be specified with the keyword. For a single layer, the *layer_name* only is used. For multiple layers, each name and corresponding layer number in the OASIS file must be specified. When OASIS DDE input is used with HDB input, the PRECISION settings of the HDB and OASIS DDE file must match.

The svrf_layer_name2 keyword is used to map layers read from vboasis_path2, svrf_layer_name3 maps layers from vboasis_path3, and svrf_layer_name4 maps layers from vboasis_path4.

- vboasis_precision_multiplier[2 | 3 | 4] {*n*[/*d*] | AUTO}

An optional argument that multiplies the PRECISION value by the supplied number *n*, and then scale the data from the file by the same amount to retain the same absolute scale as the original PRECISION setting. The default value for *n* is 1. The /*d* is used to express a rational number as a fraction (for example, a multiplier of 1.5x is declared as “3/2”).

There may be occasions when you want to re-interpret the PRECISION setting of the OASIS DDE file as specified using the vboasis_path keyword. For instance, you may want a larger PRECISION setting to allow finer shape modifications in embedded SVRF code.

If AUTO is supplied instead of a ratio, Calibre attempts to infer the correct ratio but fails if it is not a rational number, or if the combination of inferred numerator and denominator would cause arithmetic overflow. This keyword requires the use of embedded SVRF.

If vboasis_precision_multiplier2, vboasis_precision_multiplier3, and vboasis_precision_multiplier4 are specified (in conjunction with vboasis_precision_multiplier), the PRECISION scale is applied to the second OASIS input specified by vboasis_path2, vboasis_path3, and vboasis_path4, respectively.

- vboasis_layout_magnify[2 | 3 | 4] *n*[/*d*]

An optional argument that aligns the OASIS-direct-input coordinate system with the HDB coordinate system by adjusting its scale. The scale is specified as the numerator and denominator of a rational number.

This keyword is primarily used to apply embedded SVRF at the scale of the mask, but after all inverse transforms have been applied to align the MDP file inputs with the HDB coordinate system. In this case, scale up the HDB coordinate system using LAYOUT MAGNIFY and use this keyword to apply the same scale to the direct-input database.

If vboasis_layout_magnify2, vboasis_layout_magnify3, and vboasis_layout_magnify4 are specified (in conjunction vboasis_layout_magnify), the scale is applied to the second, third,

and fourth OASIS input specified by vboasis_path2, vboasis_path3, and vboasis_path4, respectively.

- `vboasis_injection[2 | 3 | 4] [-size bin_size] [-size_factor size_multiplier] [-preserve 0 | 1] [-force]`

The OASIS DDE file specified using vboasis_path may contain cells that have large amounts of data and have a large extent. These large cells degrade performance in section mode processing. To overcome this issue, cells can be partitioned into bins.

The sizes of the cells that are binned and the size of each bin (*bin_size*) is determined automatically by default when vboasis_injection is specified. You can control the bin size by providing a size multiplier using the option `-size_factor size_multiplier` (the default value is 1.0).

Alternately, the *bin_size* can be explicitly specified using the option `-size bin_size` in microns. Any cell whose extent has width or height more the *bin_size* is considered for binning.

The vboasis_injection option creates temporary data which may be reused if it is preserved using the `-preserve` option (the default is 0). Use the `-force` option to ignore any existing data.

The temporary data is read or written in a directory named *file_path.lcb* where *file_path* is argument specified to vboasis_path. The location of this directory is the same as that of *file_path*. This behavior can be changed by specifying a colon-separated list of directories using the UNIX environment variable MDPIndexSearchPath (refer to the section “Setting the Location of the Index File” in the *Calibre MDPview User’s and Reference Manual* for full details). The directories are searched in the order they are listed for reading or writing to the temporary directory.

If job deck input is used, the options apply to all OASIS files referenced by the job deck.

If vboasis_injection2, vboasis_injection3, and vboasis_injection 4 are specified (in conjunction with vboasis_injection), the options apply to the second, third, and fourth OASIS input files specified by vboasis_path2, vboasis_path3, and vboasis_path4, respectively.

- `jobdeck[2] path[-ejobdeck_clipping_severity {1 | 2}] [-ejobdeck_pattern_suffix suffix] [-data_search_path path]`

If this optional argument set is specified, input is taken directly from a MEBES-extended job deck. Such job decks may reference MEBES or OASIS chip files. The following options are available for job deck loading through the viewer:

`-ejobdeck_clipping_severity {1 | 2}` — Allows the display of data larger than the described extent in the job deck. By default, if Calibre encounters data larger than the extent, it generates an error and does not load the job deck. If set to 1, Calibre produces a warning message instead and the job deck is loaded. If set to 2, Calibre ignores the condition entirely and loads the job deck.

`-ejobdeck_pattern_suffix suffix` — Allows pattern files with different nomenclature (such as the OASIS .oas extension) to be read in MEBES extended job decks.

-data_search_path *path* — Specifies the path for a data search.

The PRECISION setting for the job deck is set to the HDB PRECISION setting. The keywords jobdeck and jobdeck2 are mutually exclusive with several other direct-input keywords:

- vboasis_path
- vboasis_precision_multiplier
- vboasis_layout_magnify

This keyword also interacts with several other MDP EMBED keywords:

- svrf_layer_name
- data_extent
- vboasis_injection
- direct_FS_access
- data_extent {HDB | FILE | FILE2 | FILE3 | FILE4 | DIRECTIN}
 - An optional argument that, when vboasis_path option is specified, uses the extent specified from one of the following options:
 - HDB — Use the cell extent from the hierarchical database (refer to “[Embed SVRF Commands in an MDP EMBED Statement](#)” on page 123 for further details on cell extents in HDBs). This is the default value.
 - FILE — Use the extent from vboasis_path.
 - FILE2 — Use the extent from vboasis_path2.
 - FILE3 — Use the extent from vboasis_path3.
 - FILE4 — Use the extent from vboasis_path4.
 - DIRECTIN — Use the extent of the logical OR of all direct inputs (vboasis_path[2]).

If you specify data_extent FILE or data_extent DIRECTIN, the extent is defined as (0,0)-(152400, 152400) um which is the full mask extent. If you only need to verify a smaller region of the mask, specify data_extent HDB and define the desired extent using an SVRF POLYGON statement to define the exact extent desired.

- index_options [-topcell *cellname*] [-parallelCblocks]
 - An optional argument that sets OASIS input module and indexing options. The -topcell option specifies the top cell name for indexing. The -parallelCblocks keyword enables parallel indexing for non-strict CBLOCK OASIS data. The cell cblocks is parsed in parallel during sequential parsing of non-strict OASIS files for faster index creation.
- oasis_output_path {-RDB | [-perSection] *path* | -INDEX [NOVIEW]}
 - An optional argument that diverts output geometry from the HDB to one or more new OASIS files. This creates a single OASIS file located at *path*, unless -perSection is

specified, in which case the output for each section is placed into a separate, syntactically complete OASIS file and no full result OASIS is created.

The -RDB keyword takes output file information from the DRC CHECK MAP statements in the embedded rule file; only a subset of DRC CHECK MAP functionality is supported. Specifically:

- Only OASIS type is allowed.
- The filename must be explicitly stated.
- The AREF, AUTOREF, MAXIMUM VERTEX, MAXIMUM RESULTS, TEXTTAG, PREFIX NAME, APPEND NAME, and PROPERTIES features are not currently supported.

All output files should be on a high-performance file system. The OASIS files have a two-level hierarchy in which the MDP EMBED sections form uniquely placed cells. The PRECISION setting of the OASIS file matches the PRECISION setting of the HDB.

The optional -INDEX [NOVIEW] keywords specify that an OASIS-type DDO database created by this command is to be indexed. The NOVIEW option generates the index faster in exchange for a slower drawing speed in the viewer.

The following example illustrates using -RDB to create multiple DDO OASIS files for different checks (enabling limited DRC CHECK MAP functionality for OASIS files only):

```
LITHO FILE eMRC [
    oasis_output_path -RDB

    <SVRFSTART>
        mrc_1 { EXT inp_1 inp_2 < 0.05 REGION } \
            DRC CHECK MAP mrc_1 OASIS 0 "output/mrc_err.oas"
        mrc_2 { INT inp_1 < 0.05 REGION }
            DRC CHECK MAP mrc_2 OASIS 1 "output/mrc_err.oas" \
            sized { SIZE inp_1 BY 0.05 } \
                DRC CHECK MAP sized OASIS 0 "output/sized.oas"
    <SVRFEND>
]
```

The following example illustrates using -perSection to create multiple DDO files for each section:

```
LITHO FILE eMRC [
    oasis_output_path -perSection output/ddo_files
    // This creates a directory "output/ddo_files" with
    // s_#.oas file for each section

    <SVRFSTART>
        sized { SIZE inp_1 BY 0.05 }
    <SVRFEND>
]
```

- `oasis_output_layers {layer [datatype] | {map_namelyear [datatype]}...}`

An optional argument that specifies the OASIS layer and data type numbers for SVRF layers written to an OASIS file. This keyword requires specification of the `oasis_output_path` keyword. There are two possible forms:

- The first form is used if the embedded rule deck produces only a single layer:

`oasis_output_layers layer [datatype]`

where `layer` is the layer name and `datatype` is the OASIS data type.

- The second form is used for multiple layers and requires the use of the MAP secondary keyword in the MDP EMBED SVRF calls:

`oasis_output_layers {map_namelyear [datatype]}...`

The map string in the SVRF call must match the `map_name` value in this keyword specification. All output layers must be mapped using this keyword.

When `oasis_output_layers` is specified in section-based processing, additional information appears in the transcript, reporting the DDO per-rule output shape counts. For example:

```
DIRECT OUTPUT RuleCheck flat1::<1> COMPLETED. Number of Results =
3375 (3375)
DIRECT OUTPUT RuleCheck flat2::<1> COMPLETED. Number of Results =
3375 (3375)
```

The `oasis_output_path` keyword supports multiple MDP EMBED blocks writing to a single OASIS file. If multiple outputs are written to a single OASIS file, then the outputs from all the blocks are added to the OASIS file. For example:

```
LITHO FILE ESVRF_1 [
//maximum_output_count 1000000000
//DDO
    oasis_output_path "./out.oas"
    oasis_output_primary HDB
    oasis_output_options -cblock -maximum_vertex ALL
    oasis_output_layers A_out 1 0 B_out 2 0 D_out 4 0
]
A_out = MDP EMBED A_in B_in FILE ESVRF_1 MAP A_out
B_out = MDP EMBED A_in B_in FILE ESVRF_1 MAP B_out
D_out = MDP EMBED A_in B_in FILE ESVRF_1 MAP D_out

LITHO FILE ESVRF_2 [
//DDO
    oasis_output_path "./out.oas"
    oasis_output_primary HDB
    oasis_output_options -cblock -maximum_vertex ALL
    oasis_output_layers C_out 3 0 E_out 5 0 F_out 6 0]
C_out = MDP EMBED C_in E_in FILE ESVRF_1 MAP C_out
E_out = MDP EMBED C_in E_in FILE ESVRF_1 MAP E_out
F_out = MDP EMBED C_in E_in FILE ESVRF_1 MAP D_out
```

Each of these separate blocks are written to a single OASIS file.

- `oasis_output_primary [HDB | DIRECTIN | FILE | -explicit topcell_name]`

An optional argument that specifies the source of the top cell name in OASIS output. The values are as follows:

HDB — The topcell name from the HDB as the topcell name of OASIS output is used.

DIRECTIN or FILE — The topcell name from the vboasis_path input is used.

`-explicit topcell_name`— The names specified by *topcell_name*.

The default setting for `oasis_output_primary` is `-explicit top`.

- `oasis_output_options [-cblock] [-maximum_vertex {mv| ALL}] [-precision val] [-magnify val]`

An optional argument that specifies options for OASIS output. With the `-cblock` option, the OASIS output is compressed using CBLOCK record. The use of `cblock` can reduce the output file size.

The maximum number of vertices (*mv*) in output polygons can be limited using the `-maximum_vertex` parameter. The default is 8192 and the symbolic value ALL corresponds to the maximum setting of 4294967295.

The `-precision` keyword specifies the precision of the OASIS output. The `-magnify` keyword is used to align the OASIS output coordinate system with the HDB coordinate system.

- `direct_FS_access [INPUT] [INTERMEDIATE]`

An optional argument that permits remote processes in a Calibre MTflex invocation to directly access a file system.

`INPUT` — Causes remote processes to directly access the argument to `vboasis_path` (including adjunct data such as injection files) and any VSB12 and OASIS files that are arguments to `input_file`.

`INTERMEDIATE` — Causes direct writing of temporary intermediate files relating to use of the `oasis_output_path` keyword.

- `mrc_only {0 | 1}`

An optional keyword that, when set to 1, reduces memory usage particularly when using small (100 um or less) section sizes. This keyword is used only for MRC flows that use MDP EMBED where the output is written to file. The default setting is 0 (off).

Note

 This keyword deactivates an internal setup code that outputs MDP EMBED data into the HDB. Without this setup code, the data is placed flat into the HDB at the top level. This does not affect MRC flows as there are typically very small amounts of output data produced. However, if the output of MDP EMBED is used in a subsequent SVRF block in the same rule deck, then the flat data can degrade the performance of those SVRF blocks.

- `mdp_remote_tmp`

An optional keyword that sets the temporary directory for Calibre MTflex processes. A single path is used for all computers used by the invocation of Calibre. The path must be visible to all computers or run time exceptions will occur. The default value is `/tmp`.

Note

 This directory should be on a file system local to each computer. If this is not done, it will introduce system performance issues.

- `<SVRFSTART>`

An optional keyword indicating the beginning of the embedded SVRF statement group.

- `svrf_statements`

Each SVRF statement must be on a line of its own. Calibre truncates any SVRF line to 1023 characters if you exceed this value.

You can specify multiple rule checks within the embedded SVRF statement group. The output of these rule checks must only be polygon data, not edge or error data. Refer to the [Calibre Verification User's Manual](#) for more information about rule checks.

You cannot specify left- or right-brackets within the embedded SVRF statement group. Therefore, any occurrence of brackets must be replaced as follows:

Table 3-1. Brackets in Embedded SVRF

Left bracket ([)	<code><LB></code>
Right bracket (])	<code><RB></code>

The PRECISION setting of the embedded SVRF statements must match that of the Calibre rule deck containing the PRECISION statement.

- `<SVRFEND>`

An optional keyword indicating the end of the embedded SVRF statement group.

Note

 The left and right angle brackets (<>) for `<SVRFSTART>`, `<SVRFEND>`, `<LB>`, and `<RB>` are literal and required.

See “[SVRF Command Support in Embedded SVRF](#)” on page 138 for a list of SVRF operations that are supported in embedded SVRF statements.

Examples

The following is a basic example of MDP EMBED:

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "sample.oas"
LAYOUT PRIMARY "TOPCELL"
DRC RESULTS DATABASE "out.oas" OASIS PSEUDO
DRC SUMMARY REPORT "cal.rep"

DRC MAXIMUM RESULTS ALL

LAYER poly 0
LAYER fill 10
LAYER marker 20

full_layer {MDP EMBED poly fill marker MAP allout FILE [
    <SVRFSTART>
        sized = SIZE poly BY 0.004 INSIDE OF marker
        all = sized OR fill
        allout {COPY all}
    <SVRFEND>
]
}

DRC CHECK MAP full_layer 9999
```

This is the MDP EMBED block.

The following shows an example of MDP EMBED with two input OASIS files:

```
LAYOUT PATH "dummy.oas" mag auto
LAYOUT PRIMARY "*"
LAYOUT SYSTEM OASIS
PRECISION 4000

DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "dummy_mdp_dde_dual.oas" oasis

LAYER dde_extent 9951
POLYGON 0 0 26653 21271 dde_extent

xor_lyr {
    MDP EMBED dde_extent MAP xor_lyr FILE my_mdp_embed
} DRC CHECK MAP xor_lyr 999

LITHO FILE my_mdp_embed [
    section_size 100

    oasis_output_path "ddo_mdp_dde_dual.oas"
    oasis_output_layers 0

// First OASIS Input
    vboasis_path "input1.oas"
    svrf_layer_name in_lyr1 50 306
    vboasis_injection -size 100
    vboasis_precision_multiplier auto

//Second OASIS Input
    vboasis_path2 "input2.oas"
    svrf_layer_name2 in_lyr2 50 306
    vboasis_injection2 -size 100
    vboasis_precision_multiplier2 auto

<SVRFSTART>
    xor_lyr { in_lyr1 xor in_lyr2 }
<SVRFEND>
]
```

The following is an example rule file that performs the XOR on two extended MEBES job decks and also performs width checks for each of the layers:

```
layout path "/mywkdr/oas_in/dummy_1000.oas"
layout primary "*"
layout system oasis
layout ultra flex yes

precision 1000

drc maximum results all
drc results database "/mywkdr/oas_out/output.hdb.oas" oasis
drc summary report "/mywkdr/summaryrep_out/output.hdb.rep"

layer dataextent 99
polygon 0 0 152400 152400 dataextent
//polygon 0 0 1000 1000 dataextent
//use a smaller extent to verify a limited region of the job deck

LITHO FILE my_mdp_embed [
    jobdeck "/mywkdr/jobdeck/jb1.ejb"
    jobdeck2 "/mywkdr/jobdeck/jb2.ejb"
    svrf_layer_name jb1_1
    svrf_layer_name2 jb2_1
    oasis_output_path "/mywkdr/oas_out/output.ddo.oas"
    oasis_output_layers width_jb1_1 11 width_jb2_1 12 xor_lay1_lay2 13
    direct_FS_access INPUT INTERMEDIATE
    <SVRFSTART>
        width_jb1_1 { INT jb1_1 < .2 ABUT < 90 SINGULAR REGION }
        width_jb2_1 { INT jb2_1 < .2 ABUT < 90 SINGULAR REGION }
        xor_lay1_lay2 { xor jb1_1 jb2_1 }
    <SVRFEND>
]

width_jb1_1 {
    MDP EMBED dataextent MAP width_jb1_1 FILE my_mdp_embed
} DRC CHECK MAP width_jb1_1 11

width_jb2_1 {
    MDP EMBED dataextent MAP width_jb2_1 FILE my_mdp_embed
} DRC CHECK MAP width_jb2_1 12

xor_lay1_lay2 { MDP EMBED dataextent MAP xor_lay1_lay2 FILE my_mdp_embed
} DRC CHECK MAP xor_lay1_lay2 13
```

SVRF Command Support in Embedded SVRF

There is a subset of SVRF operations that can be embedded into a FRACTURE, MDPVERIFY, MDP EMBED, or DENSITY CONVOLVE statement.

Table 3-2 lists the subset of the SVRF operations (otherwise documented in the *Standard Verification Rule Format (SVRF) Manual*). All other SVRF commands are not supported.

The table also indicates the commands where you must specify the secondary keyword BOUND (or RET GLOBAL BOUND set to YES) to enable the operation. All secondary keywords for this SVRF operation may not be supported in embedded SVRF. The result of this embedded SVRF operation may only match the SVRF result for data within a section and its extended range region. See the “[Influence the Interaction Range \(BOUND Keyword\)](#)” on page 143 for further details.

Table 3-2. Supported SVRF Statements for Embedded SVRF

Statement	BOUND Keyword Required?
AND — Two layers only.	No
AND — One layer variant.	Yes
ANGLE	Yes
NOT ANGLE	
AREA	Yes
NOT AREA	
COINCIDENT EDGE	No
NOT COINCIDENT EDGE	
COINCIDENT INSIDE EDGE	No
NOT COINCIDENT INSIDE EDGE	
COINCIDENT OUTSIDE EDGE	No
NOT COINCIDENT OUTSIDE EDGE	
CONVEX EDGE — If there is a WITH LENGTH constraint with an upper bound.	No
CONVEX EDGE — If there is no WITH LENGTH constraint, or if the constraint does not have an upper bound.	Yes
COPY	No
CUT — If BY NET is not specified.	Yes
NOT CUT — If BY NET is not specified.	
DEANGLE	Yes
DFM SPEC RESHAPE	No
DFM RESHAPE	
DFM SPEC VIA SHIFT	No
DFM VIA SHIFT	
DONUT	Yes
NOT DONUT	

Table 3-2. Supported SVRF Statements for Embedded SVRF (cont.)

Statement	BOUND Keyword Required?
ENCLOSE	Yes
NOT ENCLOSE	
ENCLOSE RECTANGLE	Yes
NOT ENCLOSE RECTANGLE	
ENCLOSURE	Yes
EXPAND EDGE — You cannot specify an expansion_set containing the BY FACTOR secondary keyword.	No
EXPAND EDGE — If BY FACTOR is specified.	Yes
EXTENT — The no-layer variant.	No
EXTENTS	Yes
EXTENT DRAWN	No
EXTERNAL	Yes
GROW	No
HOLES	Yes
INSIDE	Yes
NOT INSIDE	
INSIDE EDGE	No
NOT INSIDE EDGE	
INTERACT — Only if BY NET is not specified.	Yes
NOT INTERACT	
INTERNAL	Yes
LAYOUT PROPERTY AUDIT —REPLACE and APPEND are not supported.	No
LENGTH	No
NOT LENGTH	
MDP MAPSIZE	No
NOT	No
OR	No
OR EDGE	No

Table 3-2. Supported SVRF Statements for Embedded SVRF (cont.)

Statement	BOUND Keyword Required?
OUTSIDE	Yes
NOT OUTSIDE	
OUTSIDE EDGE	No
NOT OUTSIDE EDGE	
PERIMETER	Yes
RECTANGLE and NOT RECTANGLE are supported under certain circumstances, but the requirement of the BOUND keyword depends on how the constraints are defined. RECTANGLE and NOT RECTANGLE are supported if you specify at least one constraint, but you cannot specify a lower-bound constraint such as “>x”. In this case, the BOUND keyword is not required. RECTANGLE and NOT RECTANGLE are also supported if you specify only one constraint or if a constraint has no upper bound. However, in this case, the BOUND keyword is required.	Depends on constraints
RECTANGLES — If INSIDE OF is specified.	No
RECTANGLES — If INSIDE OF LAYER is specified.	Yes
RECTANGLE ENCLOSURE	Yes
SHIFT	No
SHRINK	No
SIZE — If OVERLAP ONLY is not specified.	No
SIZE — If OVERLAP ONLY is specified.	Yes
SNAP	Yes
TOUCH — If BY NET is not specified.	Yes
NOT TOUCH — If BY NET is not specified.	
TOUCH EDGE	Yes
NOT TOUCH EDGE	
TOUCH INSIDE EDGE	Yes
NOT TOUCH INSIDE EDGE	
TOUCH OUTSIDE EDGE	Yes
NOT TOUCH OUTSIDE EDGE	
VERTEX	Yes

Table 3-2. Supported SVRF Statements for Embedded SVRF (cont.)

Statement	BOUND Keyword Required?
WITH EDGE NOT WITH EDGE	Yes
WITH NEIGHBOR NOT WITH NEIGHBOR	Yes
WITH WIDTH NOT WITH WIDTH	No
XOR — Two layers only.	No
XOR — One layer variant.	Yes

Influence the Interaction Range (BOUND Keyword)

The BOUND keyword is used to enable unlimited range SVRF operations inside of embedded SVRF and, with the *value* parameter, allow user control to define the range in which the execution of the operation occurs.

Usage

BOUND [*value*]

Arguments

- *value*

An optional non-negative value specifying a range that is beyond the extent of a section in which geometrical data is included in the execution of the SVRF operation. The values for all BOUND keywords are used, in part, to determine the actual range used for all embedded SVRF operations. The default value is 0 and the BOUND *value* keyword must be the last keyword specified for each SVRF operation. Important range constraints include:

- A maximum checking range is computed and used for the general section based processing. This is the default behavior.
- The default range can be changed by using the svrf_range keyword.
- BOUND enables SVRF commands that have infinite ranges to be used within MDP EMBED.
 - These SVRF commands no longer operate with infinite range.
 - A BOUND without a value specified is = BOUND 0 = default range as defined by 1).
 - BOUND with a specified value extends the range beyond the extent of the MDP EMBED section.

Description

The SVRF operations that require the BOUND keyword are listed in [Table 3-2](#) on page 139. An SVRF operation that requires the context of the entire database to generate the correct result is classified as an unlimited range operation. For unlimited range operations to be used in section-based processing, a range needs to be specified to determine what geometrical data from outside of the extent of a section is included in the execution of the SVRF operation. The results of the unlimited range operation are accurate for the included geometrical data, but may not match the results you would get if the entire database was included. Unlimited range operations require additional consideration and understanding of their supported functionality when used in embedded SVRF.

By default, a maximum required range based on the complete set of embedded rules is automatically calculated and used for each section. You can override this calculation and set a

large fixed range than the automatically calculated range by specifying the parameter `svrf_range val`. Even with `svrf_range` specified, the `BOUND` keyword is still required to enable unlimited range operations.

Note

 Specifying an `svrf_range` value that is less than the automatically calculated range should be avoided. Failure to do so often results in generated results that do not meet the intended result of the embedded SVRF operations.

Typically, `BOUND` is declared at the end of the SVRF line. However, for the `SPACE` or `NOTCH` secondary keyword in `EXT`, `BOUND` must directly follow the word `SPACE` or `NOTCH`.

For example, the following is incorrect:

```
EXTERNAL_1LS {EXTERNAL L2 < 1.000 OPPOSITE SPACE REGION BOUND}
```

The following example is correct:

```
EXTERNAL_1LS {EXTERNAL L2 < 1.000 OPPOSITE REGION SPACE BOUND}
```

Specify a Global Bound

You can apply a “global” `BOUND` for all embedded SVRF operations by including the following keyword inside of an embedded SVRF statement group:

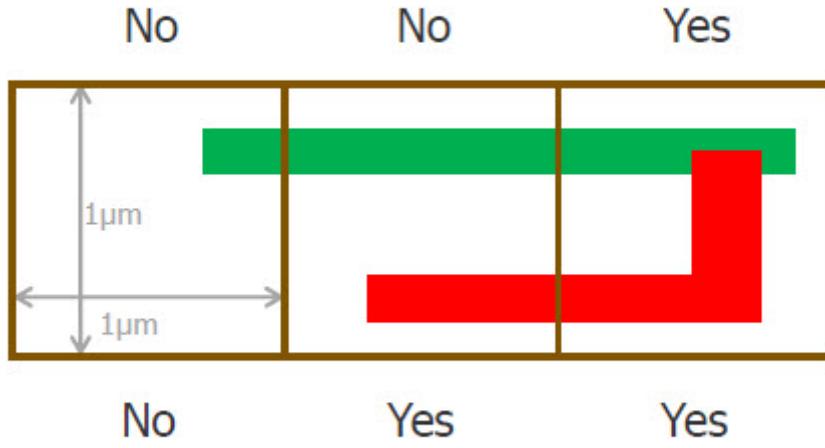
```
RET GLOBAL BOUND YES
```

When specified, this keyword implicitly supplies the `bound` secondary keyword to any embedded SVRF that would require it. Using this method, the default value for this is supplied implicitly for the `BOUND` values is 0. If a value other than 0 is desired for the `BOUND` instruction for a particular command line, the user should edit the rule file and explicitly include a `BOUND` keyword and value for those commands. The existence of the `BOUND` keyword is specified in the syntax of each SVRF that supports it.

The following figure illustrates an example where `BOUND` is used to limit the long range SVRF’s interaction range. If `BOUND` is set to 0.0, each section’s result is accurate to the section’s boundary. If `BOUND` is set to 1.0, each section’s result is accurate to 1.0 um beyond the section’s boundary.

Figure 3-4. BOUND and Long Range SVRFs

green INTERACT red BOUND 0.0



green INTERACT red BOUND 1.0

Examples

```
y = a INTERACT b BOUND  
z = c INTERACT d BOUND 0.36
```

Converting a Rule File to Embedded SVRF

If you have an existing rule file and want to implement section processing, the original rules must be split into those that go either inside or outside the embedded SVRF block.

Prerequisites

- An existing SVRF deck

Procedure

1. The process to convert a rule deck to an embedded SVRF structure is as follows:
 2. Split the original SVRF deck into two sections: a header and an embedded section.
 - a. The header contains specification and layer statements that are not allowed in the embedded SVRF section. The following SVRF commands are included in the header section:
 - o LAYOUT
 - o DRC

- FLAG
 - PRECISION
 - RESOLUTION
 - LAYER
- b. The embedded section contains variables, derivations, and one output layer per MDP EMBED or MDPVERIFY command.
- VARIABLE ...
 - *layer* = ...
 - *rule{ ... }*
 - DRC CHECK MAP (optional)

Note

 When converting a rule file to embedded SVRF, exception handling SVRF commands such as LAYOUT ERROR ON INPUT used for OASIS file indexing are not supported and are ignored during the run. Use the MDP_MISSING_REFERENCE environment variable to specify exception handling behavior instead. Refer to “[OASIS File Indexing](#)” in the *Calibre MDPview User’s and Reference Manual* for details.

3. Remove any extra outputs.

The number of output layers in the embedded section must be exactly the same number as MDP EMBED or MDPVERIFY statements) and DRC CHECK MAP statements for removed output layers must also be removed.

4. Replace the square brackets in the embedded section.
 - a. Replace all left square brackets ([) with “<LB>”.
 - b. Replace all right square brackets (]) with “<RB>”.
5. Pass the embedded output layer to a top-level rule and DRC CHECK MAP.

```
flat1 { COPY flat1 } DRC CHECK MAP flat1 101
```

6. Wrap the embedded section in an MDP EMBED command. For example:

```
flat1 = MDP EMBED layer_i MAP layer_o FILE [
    <SVRFSTART>
    include ../embedded.inc
    <SVRFEND>
]
```

Results

You should have an SVRF file converted for section-based processing.

Examples

The following is an example with an optional shell rule file showing the top-level structure of the embedded SVRF rule file:

```
include ../header.inc

flat1 = MDP EMBED layer_i MAP layer_0 FILE [
<SVRFSTART>
    include ../embedded.inc
<SVRFEND>
]

flat1 { COPY flat1 } DRC CHECK MAP flat1 101
```

Embedded SVRF With Multiple Outputs in Calibre MDPverify and MDP EMBED

You can create multiple layer and multiple file outputs in MDP EMBED and MDPVERIFY using embedded SVRF.

Procedure

1. The method to create multiple outputs using embedded SVRF is as follows:
 2. Use one MDP EMBED or MDPVERIFY command for each output layer. Use the MAP keyword if you are going to output multiple layers.
 3. Use the LITHO FILE method for specifying the commands to MDP EMBED and MDPVERIFY. If LITHO FILE is not used, then multiple layers do not run concurrently, and run times are much larger.
 4. Remove any extra DRC CHECK MAP statements from the LITHO FILE rules which are not used in a MAP layer of an MDP EMBED or MDPVERIFY command. Otherwise, an extra DRC CHECK MAP causes an error which prevents all outputs.
 5. The number of output layers must match the number of MDP EMBED or MDPVERIFY commands. Each of the output layers inside the embedded SVRF statement group must be mapped into a separate MDP EMBED or MDPVERIFY statements. Thus, the number of output layers must match the number of MDP EMBED or MDPVERIFY statements. Each of the results of the MDP EMBED or MDPVERIFY statements must be used in an operation, generally a DRC CHECK MAP to the output file. If a mismatch occurs, the following error is generated:

EMBEDDED SVRF ERROR: Too many outputs

Examples

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "simple.oas"
LAYOUT PRIMARY "TOPCELL"
DRC RESULTS DATABASE "out.oas" OASIS PSEUDO

DRC MAXIMUM RESULTS ALL

LAYER layer0 0

LITHO FILE myrules [
    <SVRFSTART>
        layer1 = layer0 SIZE BY 0.1
        layer2 = layer1 SIZE BY 0.1
        layerdi = layer2 SIZE BY 0.1
        layer1 {COPY layer1}
        layer2 {COPY layer2}
        layerei {COPY layerdi}
    <SVRFEND>
]

flayer1 { MDP EMBED layer0 MAP layer1 FILE myrules }
    DRC CHECK MAP flayer1 1
flayer2 { MDP EMBED layer0 MAP layer2 FILE myrules }
    DRC CHECK MAP flayer2 2
flayerdo = MDP EMBED layer0 MAP layerei FILE myrules
flayereo { COPY flayerdo}
    DRC CHECK MAP flayereo 3
```

MDP EMBED Output to HDB and DDO

Results from MDP EMBED can be directed to a Hierarchical Database (HDB) or directly to an OASIS file (Direct Data Output, or DDO) file using the following methods:

- Output to HDB
- Direct Database Output (DDO)
- Dual HDB and DDO

The default method is to output to the HDB, as shown in the following example:

```
drc results database "hdb.oas" oasis
out_A {
    MDP EMBED in MAP A FILE lf
}
DRC CHECK MAP out_A 1
out_B {
    MDP EMBED in MAP B FILE lf
}
DRC CHECK MAP out_B 2
LITHO FILE lf [
    <SVRFSTART>
        A { copy in }
        B { copy in }
    <SVRFEND>
]
```

In this example, both layers A and B are output to the hierarchical database file named “hdb.oas.”

The second method is to use the `oasis_output_path` and `oasis_output_layers` keywords to divert the output from the HDB to an OASIS file. The `oasis_output_path` keyword is used for directing the output geometry, and the `oasis_output_layers` keyword is used to output the layer and data type numbers for SVRF layers written to the OASIS file. This is shown in the following example:

```
drc results database "hdb.oas" oasis
out_A {
    MDP EMBED in MAP A FILE lf
}
DRC CHECK MAP out_A 1
out_B {
    MDP EMBED in MAP B FILE lf
}
DRC CHECK MAP out_B 2
LITHO FILE lf [
    <SVRFSTART>
        A { copy in }
        B { copy in }
    <SVRFEND>
oasis_output_path "ddo.oas"
oasis_output_layers A 1 B 2
]
```

As illustrated with the bolded text, the output is diverted to an OASIS file named *ddo.oas*, where layer A is mapped to layer 1 in the OASIS file, and layer B is mapped to layer 2.

The third method allows you to perform a dual HDB and direct-to-OASIS output, still using the `oasis_output_path` and `oasis_output_layers` keywords. This is illustrated in the following example:

```
drc results database "hdb.oas" oasis
out_A {
    MDP EMBED in MAP A FILE lf
}
DRC CHECK MAP out_A 1
out_B {
    MDP EMBED in MAP B FILE lf
}
DRC CHECK MAP out_B 2
LITHO FILE lf [
    <SVRFSTART>
    A { copy in }
    B { copy in }
    <SVRFEND>
    oasis_output_path "ddo.oas"
    oasis_output_layers A 1
]
```

In this example, layer A is mapped to layer 1 in the ddo.oas file (blue text). Layer B remains unmapped, and it will instead be output to HDB (the hdb.oas file) as layer 2 by default (shown in the red text).

In this dual mode, layers can only be output to either DDO or HDB (for example, output A cannot be directed to both DDO and HDB).

Currently for DDO, if a layer is unmapped then it will be flagged. For example, no map-matched output name B in `oasis_output_layers`. For DDO & HDB, if a layer is unmapped, it is output to HDB by default.

Processor Counts for MDP EMBED in the Transcript

FRACTURE and MDP EMBED operations are treated independently and may run on a different processor count. To reflect this behavior, the output transcript includes a `[DPCn]` string that is used to differentiate between the individual MDP EMBED and FRACTURE operations.

These strings appear in the licensing portion of the transcript, as well as the processor count printed at the end of the transcript. For example:

```
// Licensed Products
// -----
// Base products running on 20 cores:
// - DRC (Hierarchical)
// MDP EMBED [DPC1] products running on 8 cores:
// - MDP EMBED [DPC1]
// - MDP MASKOPT [DPC1]
// MDP EMBED [DPC2] products running on 16 cores:
// - MDP EMBED [DPC2]
// - nmMPC [DPC2]

--- CALIBRE::DRC-H LICENSING MODULE COMPLETED ...

--- CALIBRE::DRC-H COMPLETED - Thu Apr 25 11:04:33 2013
--- TOTAL CPU TIME = 0 REAL TIME = 0
--- PROCESSOR COUNT = 20
--- PROCESSOR COUNT FOR MDP EMBED [DPC1] = 8
--- PROCESSOR COUNT FOR MDP EMBED [DPC2] = 16
--- SUMMARY REPORT FILE =
```


Chapter 4

Preprocessing With SVRF

Before invoking the fracture software, you can perform various preprocessing tasks on mask data using functionality provided by the Calibre nmDRC hierarchical engine. There are several SVRF statements and operations that are most applicable to mask data preparation.

For a complete discussion of each of these statements and operations, refer to the *Standard Verification Rule Format (SVRF) Manual*.

Environment Setup	154
Layout Statements	154
Unit Statements	155
Layer Specification Statements	156
Results Reporting	156
Error Handling	157
Data Manipulation	159
Layer Selectors	159
Layer Constructors	160
Layer Creation Statements	161
Key Concepts Within SVRF	163
Calibre Results	164
The Results Database	164
The DRC Results Summary	164
The Calibre Transcript	164
Calibre Layer Data	165
Edge and Polygon Data	166
Understanding Dimension Checks	170
Understanding the Size Operation	176
Examples of Common Pre-Processing	179
Border-Aware Sizing	179
Tone-Reversal	180

Environment Setup

Specification statements in a rule file identify the layout data to be processed, provide some basic information about how it is to be processed, and specify where to store the output. The following SVRF excerpt shows some specification statements taken from a rule file. The sections that follow describe some of the statements you can use.

```
LAYOUT SYSTEM GDSII
LAYOUT PATH "gds/demo_ab.gds"
LAYOUT PRIMARY "demo"

PRECISION 1000 xxx// default values
RESOLUTION 1 xxxxxxx// default values

FLAG SKEW YES
FLAG ACUTE YES
FLAG OFFGRID YES

// The next two statements are only needed if you are saving
// fracture input layers.
DRC MAXIMUM RESULTS ALL xxx
DRC RESULTS DATABASE "gdsout/ab_out.gds" GDSII PSEUDO

DRC SUMMARY REPORT "reports/demo_ab.rep"
```

Layout Statements	154
Unit Statements	155
Layer Specification Statements	156

Layout Statements

Layout statements define the database(s) on which the Calibre nmDRC hierarchical engine operates.

Table 4-1 lists the more basic layout statements.

Table 4-1. Layout Statements Summary

Statement	Description
Layout Path	Specifies the path name for the layout design database. When multiple Layout Path statements are supplied, the first database must contain the top cell specified by Layout Primary. If you specify multiple database paths, the databases are treated as if the files were concatenated. The effect of this is that if two databases are specified and each contains a layer with the same layer number, for example layer 5, all the data on that layer in both databases is merged into a single layer before processing. Thus, all SVRF operations in the rule file operate on the combined data.

Table 4-1. Layout Statements Summary (cont.)

Statement	Description
Layout Primary	Required: Identifies the top level or cell on which to operate.
Layout System	Required: Identifies the database format of the layout design database.
Layout Magnify	Specifies that the input layout database is to be magnified by the given value as it is being read into the Calibre application. When specified, all coordinate data, including placement base points and array pitches, is multiplied by the given value, regardless of its hierarchical position.
Layout Window	Specifies a polygon window that defines the portion of the design to be affected by subsequent RuleChecks. This statement allows you to perform pre-processing only on the data that you intend to fracture. Note that Layout Window is applied before Layout Magnify.

In addition to these required statements, to obtain the best performance from the Calibre nmDRC hierarchical engine, it is recommended that you supply an optional statement ([Layout Base Layer](#) or [Layout Top Layer](#)) to identify the layout base layers. For more information, refer to the *Calibre Post-Tapeout Flow User's Manual*.

Unit Statements

Unit statements provide information about the database unit precision and resolution.

[Table 4-2](#) lists the more commonly used unit statements.

Table 4-2. Unit Statement Summary

Statement	Description
Unit Length	Specifies the user unit for length. When a number is supplied, it specifies the user unit of length in terms of meters. When a factor is supplied, it specifies the user unit in terms of another unit, such as mil, mm, cm, or inch. The default is 1E-6, or 1 micron.
Precision	Defines the ratio of database units to user units. Precision can be specified as a single value or by two integers that define a ratio. The default is 1000. Given the default unit length of 1 micron, and the default PRECISION of 1000, the default database unit is one nanometer.
Resolution	Defines the layout grid step size. When only one value is supplied, the grid step size is the same in both the x and y direction. The default value is 1.

Layer Specification Statements

Layer specification statements make data in the design database available to the Calibre nmDRC hierarchical engine. During any run, the application ignores all data except the layers specified using these statements.

[Table 4-3](#) lists the layer specification statements.

Table 4-3. The Layer Specification Statements

Statement	Description
Layer	<p>Declares a layer in the design database, making it available within the Calibre nmDRC hierarchical engine. The declaration references a database layer by number and assigns it a name by which it is referenced within subsequent operations.</p> <p>In most cases, the layer you declare already exists in the design database. By declaring a layer that does not exist in the database, you create an empty original layer. This is important, as some operations only work with original layers.</p> <p>The layer name is a user-defined alphanumeric string. It cannot be an SVRF reserved word, unless it is surrounded by single ('') or double ("") quotes. Calibre names must be unique within the SVRF file.</p> <p>You can assign the original layer as many names as needed. However, you can only assign the Calibre name to one original layer or group of original layers.</p>
Layer Map	<p>Creates a new “original” layer by mapping data of the specified datatype(s) on the specified layer(s) to a new layer.</p>
Layer Directory	<p>You should use this statement only when you are inputting a large, flat design that could exceed your memory limits.</p> <p>Specifies a path name in which Calibre stores geometric data used during the Calibre nmDRC-H run. This disk-based storage is an alternative to memory-based storage, which can save significant amounts of memory for the FRACTURE operations. Calibre deletes the generated data and directory upon completion of the Calibre nmDRC-H run.</p>

Results Reporting

DRC statements let you specify how the Calibre nmDRC hierarchical engine treats the results of SVRF operations.

[Table 4-4](#) lists the more commonly used DRC statements.

Table 4-4. Calibre nmDRC Statements

Statement	Description
DRC Maximum Results	Specifies the number of results to save to the results database. The default value is 1000. For saving fracture input, this statement must set DRC Maximum Results to ALL to ensure that the application saves all corrected geometries. This statement is not related to saving fractured data.
DRC Results Database	Specifies the full pathname to the primary results database, and its format. The optional keyword PSEUDO specifies that the results database should include the additional levels of hierarchy that the application creates.
DRC Summary Report	Specifies the full pathname for the summary report file. Additional optional arguments let you control how this file is saved.
Layout Error On Input	Specifies whether errors or warnings encountered while reading in a layout database result in fatal errors.
(RuleCheck Statement) name {layer_operation layer_definition ... [layer_operation layer_definition] }	Instructs the application to output the results of the specified layer operations to the results database.
DRC Check Map	Instructs the application to output the results of the specified RuleCheck on the specified layer in the results database. You can also use this statement to output the RuleCheck to a database other than the primary results database. To take advantage of this functionality, you must specify the pathname and format of the database.
DRC Magnify Results	Specifies that the DRC results database, as it is being output by the Calibre nmDRC application, is to be magnified by the given value.
DRC Results Database Precision	Specifies the actual PRECISION value in the DRC results database. It has no effect on coordinate values output to the database.

Error Handling

Flag statements let you control how the Calibre nmDRC hierarchical engine responds to certain types of errors.

[Table 4-5](#) lists the more commonly used flag statements.

Table 4-5. Flag Statements

Statement	Description
Flag Acute	Instructs the application to issue a warning when it encounters an acute angle.
Flag Nonsimple	Instructs the application to issue a warning when it encounters unmerged, original, polygons that are not orientable or non-simple (containing portions that overlap other parts of themselves).
Flag Nonsimple Path	Instructs the application to issue a warning when it encounters a path that is non-simple (overlaps itself at some point).
Flag Offgrid	Instructs the application to issue a warning when it encounters an off-grid vertex.
Flag Skew	Instructs the application to issue a warning when it encounters a skew edge.

Data Manipulation

Layer operations and layer creation statements isolate specific layer data and modify that data to generate new data. Some operate on a single layer, some on a pair of layers. When deciding which operation to use, think in terms of what each operates on and what it does with that data.

Each operation operates on either polygons or edges. (For information on Calibre layers, and polygon and edge data, refer to “[Calibre Layer Data](#)” on page 165.) The resulting data consists of only one of these types of data. Because of this, you must pass the correct type of data to the operation.

Note

 To the Calibre nmDRC hierarchical engine, edge data is more than just isolated line segments. Each edge maintains a record of the polygon from which it was derived, as well as which side is interior and which is exterior.

Each operation can do one of two things. If it is a Layer Selector, it can select existing data from specified input layer(s). If it is a Layer Constructor, it can create new polygon or edge data by modifying existing data.

Layer Selectors	159
Layer Constructors	160
Layer Creation Statements	161

Layer Selectors

Layer selectors are operations that select existing data that meet the specified criteria. They may select data from one layer or from all specified layers. Some select polygons; others select edges.

Most operations are layer selectors. [Table 4-6](#) describes a few of the more commonly used layer selector operations.

For a complete list of layer operations, refer to the [Standard Verification Rule Format \(SVRF\) Manual](#).

Table 4-6. Commonly Used Layer Selector Operations

Operation	Description
Area	Selects all polygons on layer1 having a total area that satisfies the specified constraint.
Copy	Copies (selects) all data on the specified layer.
Inside	Selects all polygons on layer1 that lie inside polygons on layer2.
Outside	Selects all polygons on layer1 that lie outside polygons on layer2.

Table 4-6. Commonly Used Layer Selector Operations (cont.)

Operation	Description
Length	Selects all edges on the specified layer that satisfy the length constraint.
Interact	Selects all layer1 polygons that either share some or all of their area with a layer2 polygon, or are outside all layer2 polygons, but have a coincident outside edge or edge segment with a layer2 polygon.
Internal ¹	Defines a required separation between the interior sides of two edges.
External ¹	Defines a required separation between the exterior sides of two edges.
Enclosure ¹	Defines a required separation between the interior side of an edge on one layer and the exterior side of an edge on another layer.
Convex Edge	Selects edges based on the specified constraints, which can include the value of the angle at one or both ends of the edge, the length of the edge, or the length of abutting edges.
Coincident Edge	Selects all layer1 edges or edge segments that are coincident with layer2 edges.

1. Internal, External, and Enclosure are dimension checks that you can use as layer selectors or layer creators.

Layer Constructors

Layer constructors are operations that create new data by modifying existing data. Some construct polygon data from existing polygons. Others create polygon data from existing edges.

Table 4-7 describes a few of the more commonly used layer constructor operations. For a complete list of operations, refer to the [Standard Verification Rule Format \(SVRF\) Manual](#).

Table 4-7. Commonly Used Layer Constructor Operations

Operation	Description
AND	Creates new polygon data from all polygon regions that are common to one or more polygons on layer1 and one or more polygons on layer2.
OR	Creates new polygon data by combining all polygons on layer1 and all polygons on layer2, merging overlapping polygons into larger polygons.
NOT	Creates new polygon data from all polygon regions on layer1 that are not common to any polygons on layer2.

Table 4-7. Commonly Used Layer Constructor Operations (cont.)

Operation	Description
XOR	Creates new polygon data from all polygon regions on layer1 or layer 2 common to exactly one polygon on either layer 2 or layer2. (That is, all polygon regions that do not overlap any other polygon region.)
Density	Creates a derived polygon layer containing rectangles that define those areas of the layer that have a density that meets the specified constraint. For this operation, density is defined as the ratio of the area contained in polygons to the area of the rectangle being examined. You can also use the density operation to measure the density in a predefined area.
Expand Edge	Expands edges into rectangles. The expansion parameters define how the application expands the edge.
Extent	Generates a derived polygon layer consisting of one rectangle that equals the database extent (also called the boundary or bounding box).
Grow	Expands polygons on the input layer in the direction of the x-axis, y-axis, or both.
Size	Expands or shrinks polygons according to the sizing parameters. The sizing parameters define the inward or outward distance by which the polygons are sized. They can also define bounding areas and specify how the application treats the resulting polygons (such as, merge polygons, output overlaps only, or truncate).
Shrink	Contracts polygons on the input layer in the direction of the x-axis, y-axis, or both.
Snap Offgrid	Snaps all offgrid geometries to the grid specified in the Resolution specification statement or the appropriate Layer Resolution specification statement, if present.
Snap	Snaps each vertex on the input layer to the grid specified by the resolution parameter(s).

Layer Creation Statements

Layer creation statements let you store the results from a layer operation as a derived layer, which you can reference in other layer operations.

[Table 4-8](#) lists the layer definition statements.

Table 4-8. The Layer Definition Statements

Statement	Description
Layer Definition Statement layer_name = layer_operation	Creates a derived layer with the specified name. The contents of this layer are the results of the operation.

Key Concepts Within SVRF

There are a number of SVRF key concepts that apply for mask data preparation operations.

Calibre Results	164
The Results Database	164
The DRC Results Summary	164
The Calibre Transcript	164
Calibre Layer Data	165
Edge and Polygon Data	166
Understanding Dimension Checks	170
Understanding the Size Operation	176

Calibre Results

Each FRACTURE operation generates an output file in the proper format and does not return any data to the Calibre nmDRC hierarchical engine. Because the data is not returned to the Calibre engine, the MEBES or JEOL output is not considered to be Calibre results. This does not mean that Calibre results are not relevant to fracturing. There are several types of results generated by the Calibre nmDRC Hierarchical engine.

The types include:

- Results Database
- Calibre Transcript
- DRC Results Summary

The Results Database

This database stores the results of Calibre operations performed either before or after invoking the Calibre FRACTURE software.

The Calibre nmDRC hierarchical engine writes results to the database file specified with the **DRC RESULTS DATABASE** statement in the SVRF rule file. If this is a GDSII or OASIS database the Calibre nmDRC hierarchical engine writes all the data to layer 0. Using the DRC CHECK MAP statement, you can assign different types of data to different layers and/or instruct the application to write the results to different database files.

The DRC Results Summary

When you specify a pathname using the DRC SUMMARY REPORT statement, the Calibre nmDRC hierarchical engine generates a results summary text file and saves it to this file. The file contains heading information, describing the particulars of the run, runtime warnings list, statistics for the layers before and after processing, and a runtime summary.

The Calibre Transcript

The Calibre nmDRC hierarchical engine automatically generates a text transcript at runtime and sends the transcript to stdout, which is typically the shell window from which you invoked the application. During the Calibre run, the transcript displays event logs, warning messages, and summary information. This transcript documents the tasks the application performs: compiling the rule file, loading in the design data, and operating on that data. It is good practice to save the

transcript by redirecting the output to a file, which you can do when you invoke Calibre, by using the pipe/tee syntax, as illustrated:

```
calibre -drc -hier -turbo -turbo_litho fracture.drc | tee fracture.log
```

For a more complete description of the Calibre transcript, refer to the [Calibre Verification User's Manual](#).

Calibre Layer Data

The Calibre nmDRC hierarchical engine distinguishes between four types of layers:

- Original layers (also called drawn or design layers)
- Derived polygon layers
- Derived edge layers
- Derived error layers

Original layers (or drawn layers) are layers that represent original layout data. In the layout database file, you reference original layers by number. In a rule file, you assign a layer name to each original layer that you intend to use, then reference it by its name.

To assign a layer name to an original layer, you must declare it using the Layer statement. This statement makes all data on the original layer available within the application. If the layer does not exist in the layout database, the statement “creates” an empty original layer.

If you need to access only certain types of data on a layer, you can use the Layer Map statement to “create” a new original layer containing only this data. You must specify the datatype(s) and the layer or layers containing the data, and assign it a layer number that does not exist in the layout database. You can then make that data available by declaring the new layer using the Layer statement.

Derived polygon layers contain polygons generated as the result of layer operations, such as Boolean functions, area functions, and polygon-directed dimension check operations.

Derived edge layers contain edges or sub-edges of polygons generated as the result of layer operations, such as edge operations and edge-directed dimension check operations.

Derived error layers represent clusters of one, two, three, or four edge segments. Each cluster is the result of an error-directed dimensional check operation. You cannot use error layers for any purpose other than reporting errors that exist between the edges within each cluster. For more information on error layers, refer to the [Calibre Verification User's Manual](#).

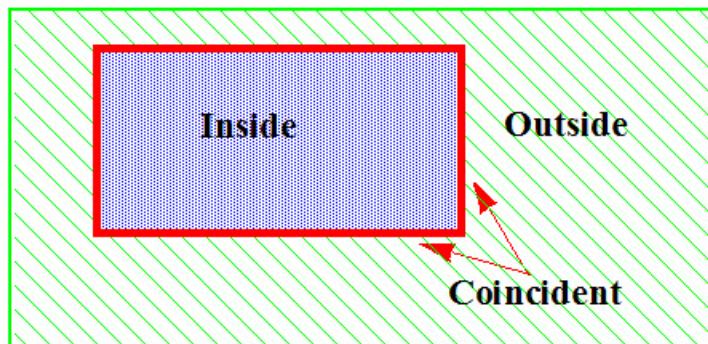
Edge and Polygon Data

Each Calibre SVRF operation operates on either polygon data or edge data or both. As you classify and manipulate data with SVRF, you must pay close attention to the type of data each operation returns: edge layers or polygon layers.

Polygon data consists of whole polygons. In most cases, the Calibre nmDRC hierarchical engine automatically merges the polygon data as part of any polygon operation. That is, if any polygons on a single layer overlap or share an edge, the Calibre nmDRC hierarchical engine merges them into one polygon. Merged data is normally a more accurate depiction of the true mask than unmerged data.

Each polygon divides space into three categories: inside, outside, and coincident. In [Figure 4-1](#), imagine that the space extends to the edge of the mask, in all directions. The red rectangle represents the polygon itself and the portion of the space that is coincident with it. The blue area shows the portion of space that is inside; the green striped area shows the space that is outside.

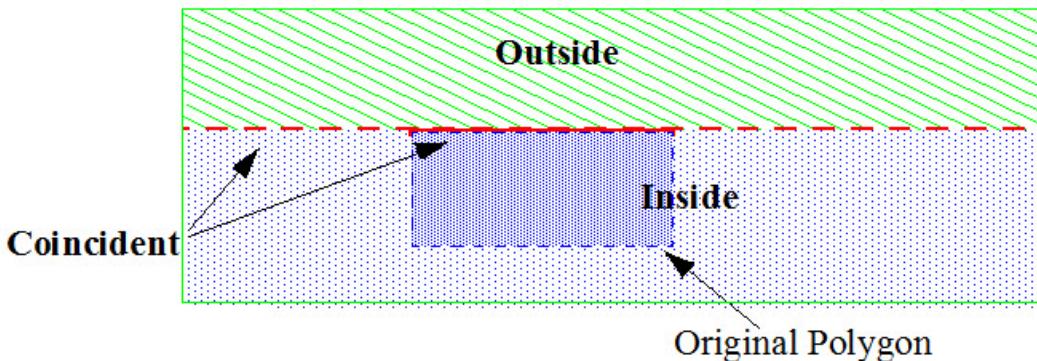
Figure 4-1. How Polygon Data Divides Space



Edge data consists of individual polygon edges, or portions of edges. These edges contain both geometric data and polygon reference data. Geometric data refers to the x and y coordinates of the edge. Reference data refers to a record of the polygon to which the edge belongs and its orientation (which direction is inside, which is outside.)

Edge data also divides space into three categories: inside, outside, and coincident, however the meanings are slightly different. All points on the side of the line that was inside of the original polygon are on the inside of the edge. All points on the side of the line that was outside of the original polygon are on the outside of the edge. All points colinear with the line are said to be coincident with the edge. In [Figure 4-2](#), the small rectangle drawn with a thin dotted line is included to show how the orientation of the edge relates to the original polygon. Note that the inside of the polygon is a small subset of the inside of the edge.

[Figure 4-2](#) shows the edge drawn as a thick line, with a dashed line drawn through the edge and extending beyond it in both directions.

Figure 4-2. How Edge Data Divides Space

Layer of Origin..... **167**

Layer of Origin

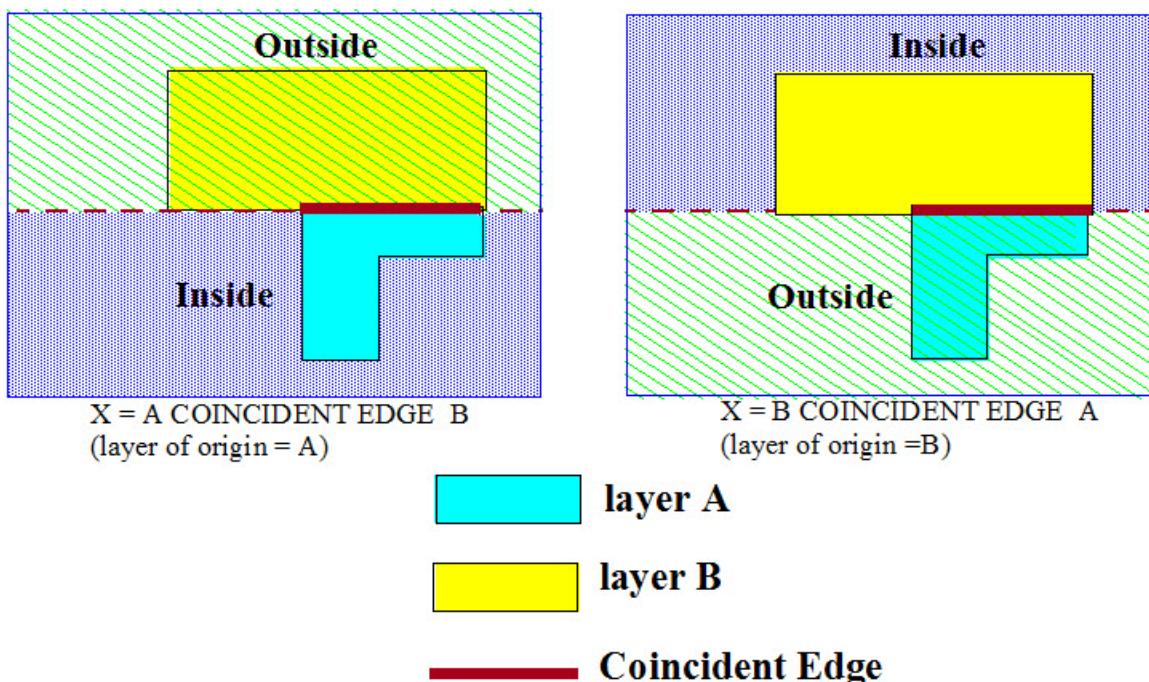
Because polygon reference data associates each edge with a specific polygon, you must pay close attention to the layer of origin, that is, where the data comes from. For example, it might appear that the following layer definitions produce the same data in the layer X:

Table 4-9. Layer of Origin Example

<pre>X = A COINCIDENT EDGE B X = B COINCIDENT EDGE A</pre>	<pre>// Edges on layer A that are coincident // with edges on layer B. // Edges on layer B that are coincident // with edges on layer A.</pre>
--	--

However, the polygon reference data stored with the results is different depending on the layer of origin. While both operations appear to generate the same geometric data, one contains edges selected off layer A, and the other contains edges selected off layer B. In [Figure 4-3](#), inside and outside are different depending on the layer of origin.

Figure 4-3. Reference Data Dependence on Layer of Origin

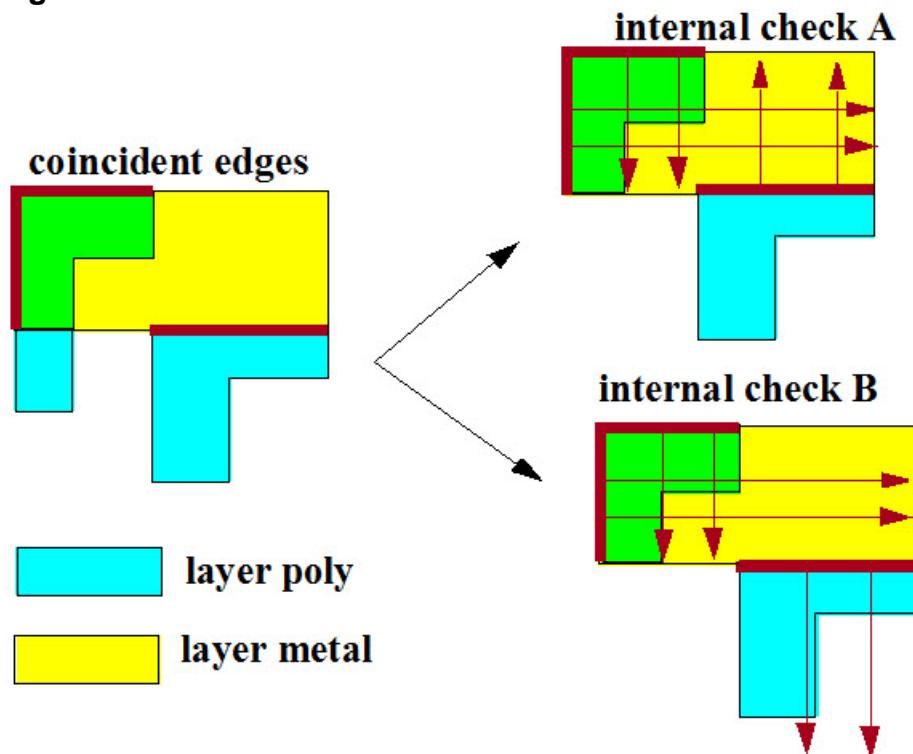


Dimension checks, which measure distances in a specific direction relative to the edge, provide a good example of just how important the layer of origin is. For example, the INTERNAL operation in the following sequence A, has an entirely different effect than the INTERNAL operation in the sequence B, which follows it:

<pre>X = metal coincident edge poly internal X metal < 3</pre>	<pre>// Edges on layer metal that are // coincident with edges on layer poly. // Find internally facing edges closer // than 3 microns.</pre>
<pre>X = poly coincident edge metal internal X metal < 3</pre>	<pre>// Edges on layer poly that are // coincident with edges on layer metal. // Find internally facing edges closer // than 3 microns.</pre>

In sequence A, layer X contains geometric data carrying polygon reference data from layer metal, while in sequence B, this data carries polygon reference data from layer poly. [Figure 4-4](#) shows how this results in the internal check, which looks inside from edge X, looking in opposite directions for the lower coincident edge.

Figure 4-4. The Effect of Reference Data on Dimension Checks



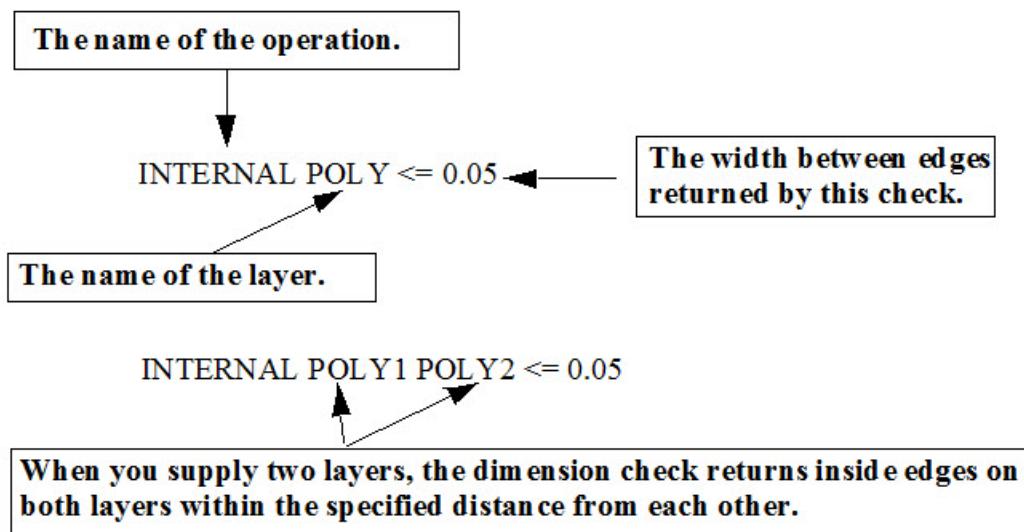
Understanding Dimension Checks

Dimension checks are the most commonly used types of SVRF operations when performing rule checks. These operations measure distances between opposing edges. Because dimension checks allow you to classify edges based on key properties, such as width and spacing, they serve as the foundation for all RET-related processing.

The Calibre nmDRC hierarchical engine provides three dimension checks:

- INTERNAL — Defines a minimum separation between the interior sides of two edges.
- EXTERNAL — Defines a minimum separation between the exterior sides of two edges.
- ENCLOSURE — Defines a minimum separation between the interior side of an edge on one layer and the exterior side of an edge on another layer. This results in returning edges on one layer that are enclosed by polygons on the other layer.

The following example shows two simple dimension checks. Using additional arguments, you can fine-tune the dimension check to return only the data you are interested in. For more information on dimension check arguments, refer to the *Standard Verification Rule Format (SVRF) Manual*.



The Calibre nmDRC hierarchical engine performs dimension checks as a three-step process:

1. Identify appropriate edge pairs.
2. Construct measurement regions.
3. Return those portions of an edge that intersect the measurement region for the opposite edge.

The sections that follow describe each step.

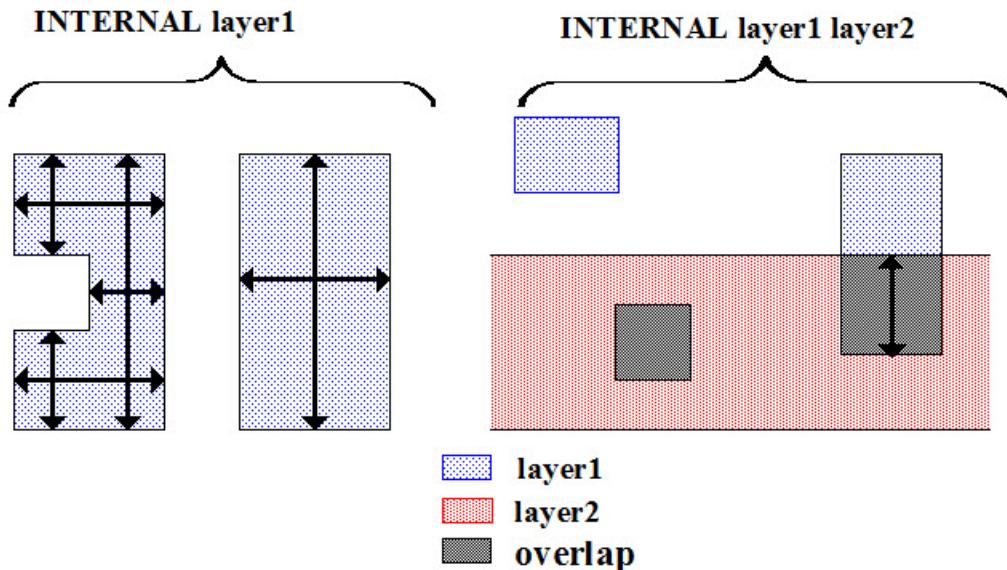
Identify Edge Pairs	171
Create Measurement Regions.....	173
Return Intersection Edges.....	175

Identify Edge Pairs

The Calibre nmDRC hierarchical engine begins the measurement process by evaluating all the edges on the layer (or pair of layers) to find “appropriate” edge pairs, that is, all pairs of edges that meet the criteria for the operation. The first criterion relates to the orientation of the edges. Each dimension check measures from a specific side of the first edge to a specific side of the second edge:

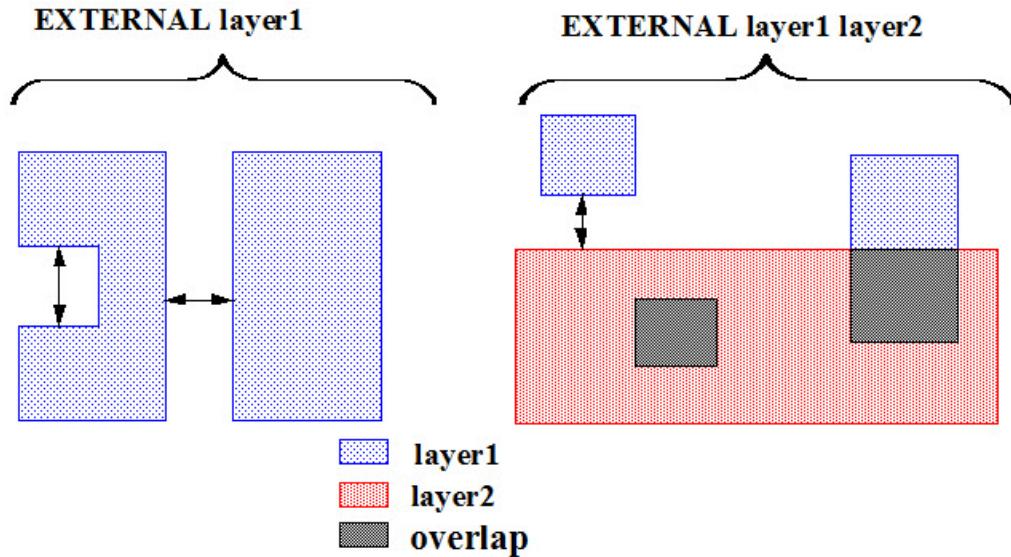
- For the Internal dimension check, the application measures from the inside of one edge to the inside of another edge.

Figure 4-5. Edge Pairs Measured by the Internal Operation



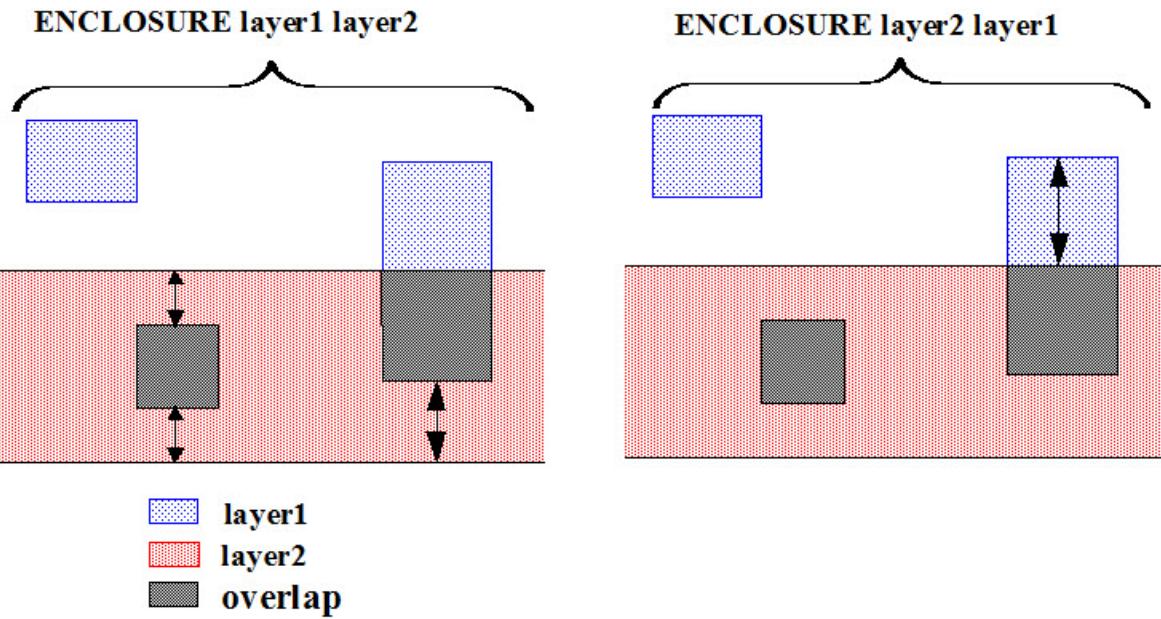
- For the External dimension check, the application measures from the outside of one edge to the outside of another edge.

Figure 4-6. Edge Pairs Measured by the External Operation



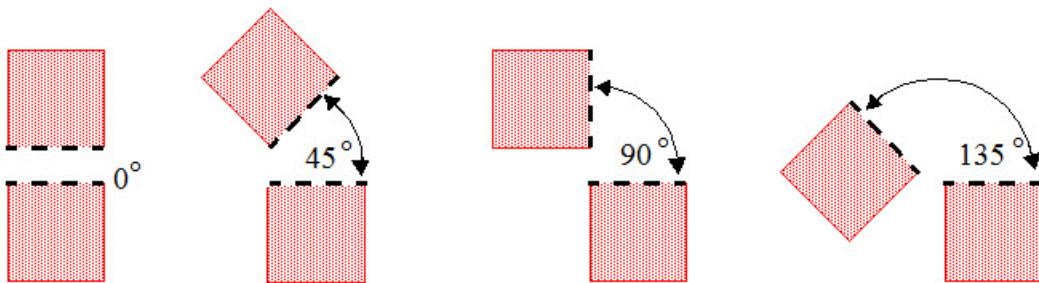
- For the Enclosure dimension check, the application measures between the outside side of an edge on the first input layer to the inside of an edge B on the second input layer.

Figure 4-7. Edge Pairings Measured by the Enclosure Operation



The second criterion for “appropriate” edge pairs relates to the angle between the specified sides of the edges. [Figure 4-5](#) through [Figure 4-7](#) shows edge pairings that measure the distance between parallel edges (angle = 0). By default, the application considers a pair of edges to be appropriate if the angle between the two edges ≥ 0 and < 90 . You can use additional arguments with dimension checks to override this default behavior; however, the angle must always be less than 180 degrees.

Figure 4-8. Measuring “Appropriate” Angles for Edge Pairings



For the EXTERNAL dimension check, the appropriate angle is between the outsides of the dashed edges.

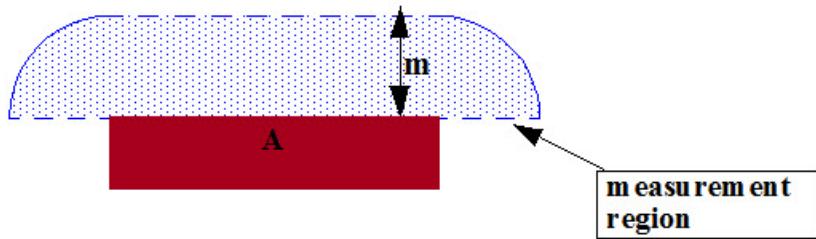
Create Measurement Regions

Once the Calibre nmDRC hierarchical engine has identified pairs of edges that meet the criteria, it creates measurement regions for each of the edges in a pair. The application uses measurement regions to measure the separation between edges. Each measurement region extends in the direction of the dimension check, either outside or inside. It contains all points in that direction such that the distance from the edge satisfies the constraint for the operation.

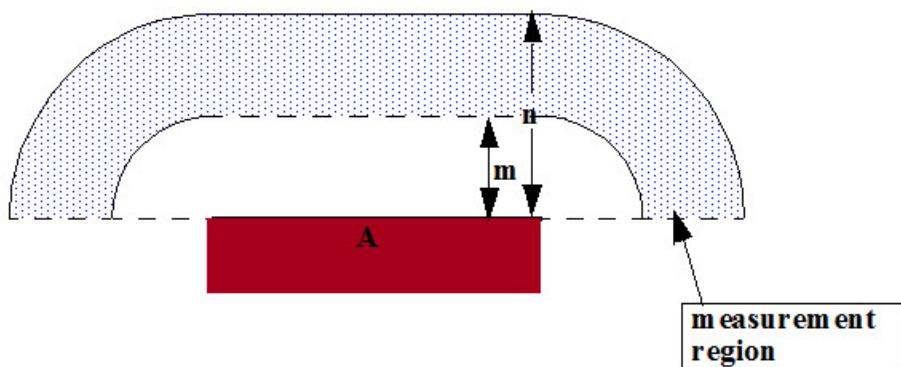
Figure 4-9 shows two regions outside of edge A. The first region assumes the constraint $x < m$ and the second region assumes the constraint $m < x \leq n$. By definition, the measurement region does not contain the points on the line containing edge A.

Figure 4-9. Measurement Regions for Edge A

rule1 { external layer1 < m }



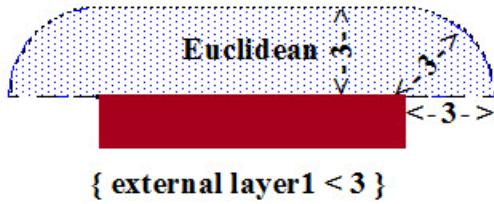
rule2 { external layer1 > m < n }



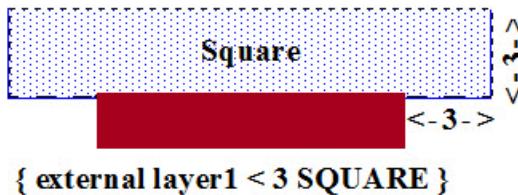
Metrics

Note the rounded shape of the measurement regions in [Figure 4-9](#). You control the shape of the measurement regions by specifying the metric, or measurement style. The Calibre nmDRC hierarchical engine supports five metrics:

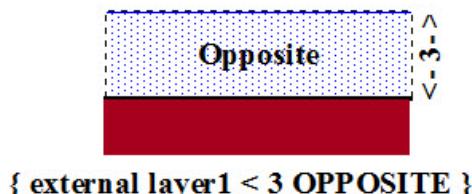
- Euclidean. The Euclidean metric forms a region with rounded boundaries that extend past the corners of the selected edges. This is the default metric.



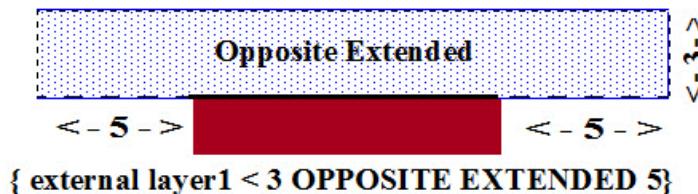
- Square. The square metric forms a region with right-angle (90 degrees) boundaries that extend past the corners of the selected edges.



- Opposite. The opposite metric forms a region with right-angle boundaries that do not extend past the corners of the selected edges.



- Opposite Extended. The opposite extended metric forms a region with right-angle boundaries that can extend past the corners of the selected edges by a value you specify.



- Opposite Symmetric. The opposite symmetric metric uses the opposite metric with post-processing for use with non-parallel edges. For more information on this metric, refer to the [Calibre Verification User's Manual](#).

Return Intersection Edges

The final step in performing the dimension check returns those portions of the edges that intersect the measurement region for the opposing edge. Because the measurement region for edge A does not contain the line containing edge A, dimension checks do not return any existing points of intersection.

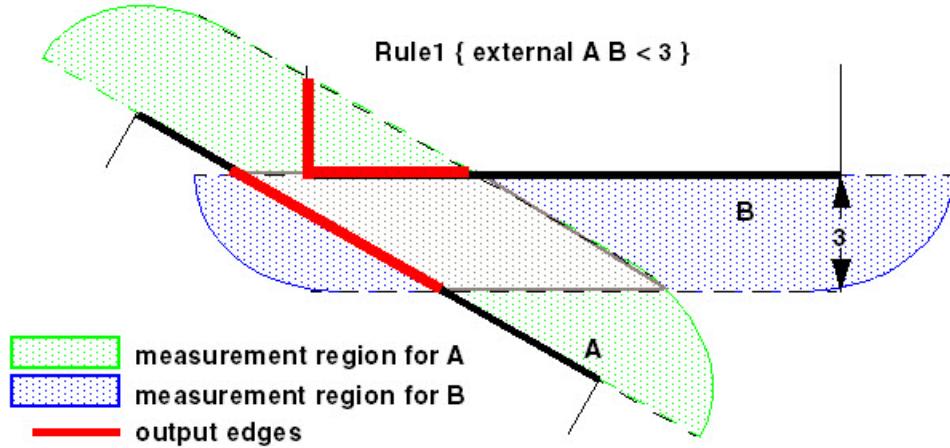
Example

Assume the measurement is from the outside of edge A to the outside of edge B and that the operation instructs the application to return all edges closer than 3 microns. Using the default metric, which is Euclidean, the operation begins processing by constructing two regions. One

region consists of all points in the half-plane on the outside of edge A that are within 3 microns of edge A; a similar region consists of all such points around edge B.

The output is an edge pair consisting of the portion of edge A that intersects the measurement region for edge B and the portion of edge B that intersects the measurement region for edge A.

Figure 4-10. Creating Measurement Regions



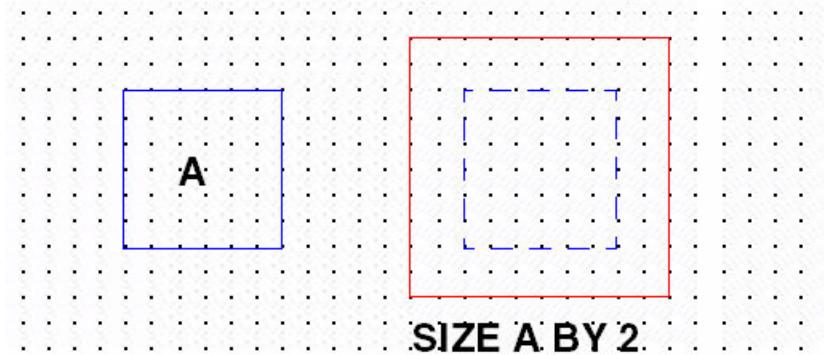
Understanding the Size Operation

When preparing mask data for fracturing, you are very likely to need to size the data. Depending on the tool you are using, you may be used to giving the size value per shape, or per edge. With the Calibre SIZE operation you specify the size value per edge. The sizing value must be an integer multiple of the PRECISION defined in the rule deck used to perform the sizing.

In its most simple form, this operation oversizes or undersizes the polygons on the specified layer. Some things to keep in mind are:

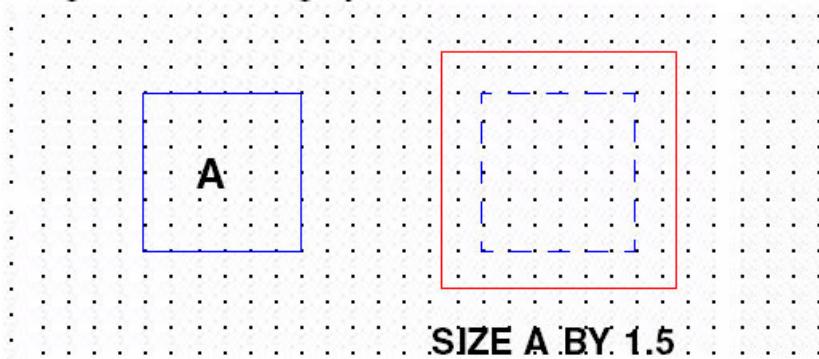
- The SIZE operation sizes each edge, rather than the entire polygon. That is, each edge is moved by the SIZE BY amount. [Figure 4-11](#) shows how this affects the final output. Notice that SIZE BY 2 causes polygon A to become 4 units wider than before sizing. To size an entire polygon by N using the SIZE operation, you must therefore size each edge by (N/2).

Figure 4-11. Simple Oversizing



- Sizing a polygon by an odd multiple of address units results in the polygon vertices being drawn off the address-unit grid. [Figure 4-12](#) shows the sizing required to obtain a polygon that is 3 units larger than the original, which is SIZE BY (3/2). Notice the vertices are now off-grid relative to the address-unit grid. However, because the Calibre FRACTURE operation automatically snaps off-grid data onto the grid, and because the SIZE operation affects all geometries on the layer equally, the dimensions and spaces between structures are preserved.

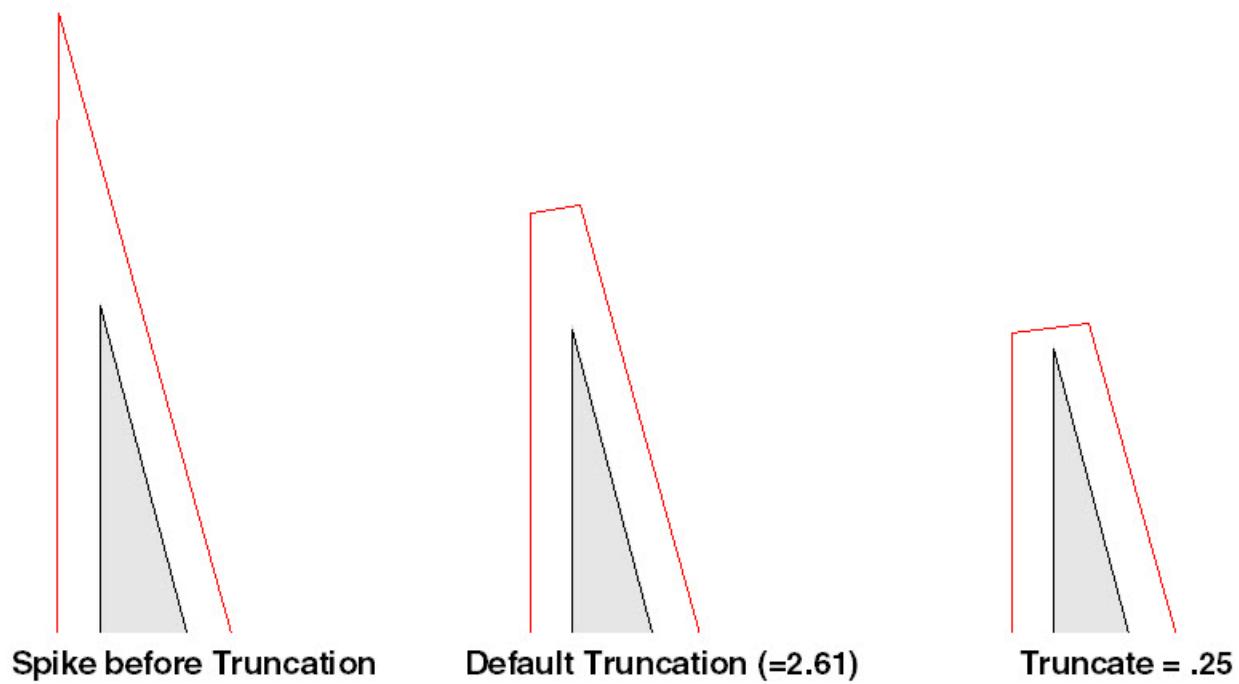
Figure 4-12. Sizing by an Odd Number of Grid Units



For some special cases, such as those involving border-aware sizing, grid snapping after sizing by an odd multiple of address units can introduce some complexity. Refer to Appendix A, “[Odd-Multiple Grid Sizing](#)” of this document for a detailed discussion of this situation.

- When mask data contains angles that are smaller than 45 degrees, oversizing can lead to unexpectedly large spikes. By default, the SIZE operation truncates these spikes so that they extend the same distance beyond the original polygon that a 45 degree angle would extend. You can control exactly how far these spikes extend by defining a TRUNCATE distance. During processing, the application allows the spike to extend beyond the original geometry by a distance equal to the TRUNCATE value times the distance by which the polygon is being sized. [Figure 4-13](#) shows the effect of two different truncation values.

Figure 4-13. Truncating Spikes



Examples of Common Pre-Processing

There are a number of examples of real-life mask data pre-processing tasks, including examples of how you can perform these tasks using SVRF.

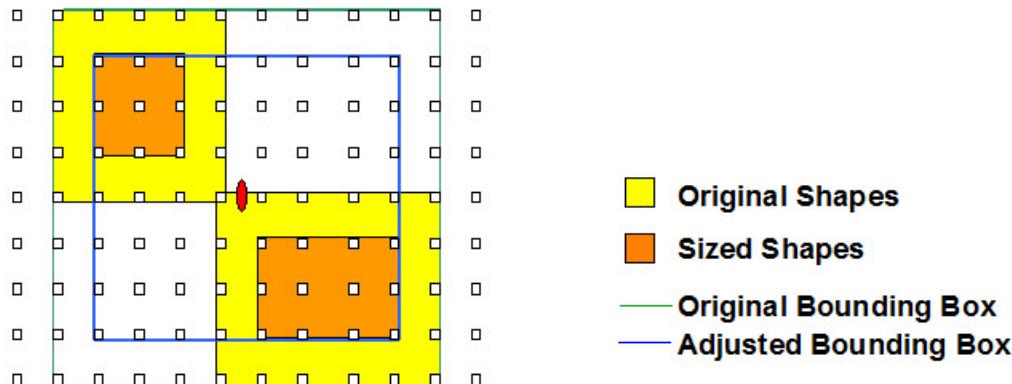
Border-Aware Sizing	179
Tone-Reversal	180

Border-Aware Sizing

Both the Size, Shrink, and Grow operations in SVRF perform simple sizing on the entire fracture region in which the border (or bounding box) is sized as well.

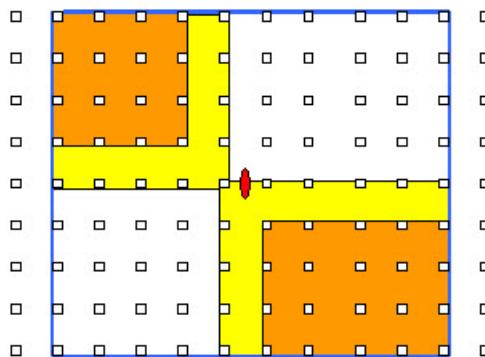
This is illustrated in [Figure 4-14](#).

Figure 4-14. Simple Shrink



Because data assembly in the job deck requires that different pieces come together in a controlled way, it is often necessary to preserve the border size while preserving the edge/border relationships for any geometries that touch the border.

Figure 4-15. Border-Aware Shrink



The following SVRF example performs this border-aware shrinking. If you choose to use this code in your own SVRF, be sure to replace the value (0.03) with the actual shrink size.

```
Layer in_layer 1
border = EXTENT in_layerxxxx// returns bounding box of input data
border_ring = (SIZE border BY 0.03) NOT border
sized_layer = SIZE (in_layer OR border_ring) BY -0.03
```

Tone-Reversal

Reversing the tone of a mask is a common task that must be performed properly in order to guarantee mask fidelity. You can perform tone reversal either using SVRF (before fracturing or in embedded SVRF) or as part of the fracture process, using the reverse_tone keyword.

When performing tone reversal using SVRF, the key is ensuring that the portion of the layer that you invert is at least as large as the region you intend to fracture. In the following code example, it is equal to the fracture region.

```
Layer in_layer1 1
Layer new_layer 100 // create an original layer not in database
Polygon x1 y1 x2 y2 new_layer // polygon with coordinates of region
                                // to_fracture
to_fracture = new_layer NOT in_layer1 // inverse of geometries
Fracture JEOL to_fracture INSIDE OF LAYER new_layer FILE [
```

Chapter 5

FRACTURE Examples

There are a number of examples available that illustrate proper usage of the FRACTURE operation.:

MEBES Example Using Magnify and Rotate	181
JEOL Example.....	181
VSB11 Example	182
HITACHI Example	184
Micronic Example	185
OASIS.MASK Example	186

MEBES Example Using Magnify and Rotate

The following is an example MEBES fracture operation.

```
// MEBES MODE 5 FRACTURE
fracture1 {FRACTURE MEBES orig6 INSIDE OF (-5000) (-5000) 5000 5000 FILE [
    mode 5
    address_size 0.02
    compaction_stripes 64
    magnify 4
    rotate 90
    reverse_tone
    computation_mode section
    file_name TEST1XXXX.PF
    log_file_name mebes_fracture.log
]
}
```

MEBES Output

A single MEBES pattern file in the current working directory:

TESTXXXX.PF

JEOL Example

The following illustrates an example JEOL fracture operation.

```
fracture3 {FRACTURE JEOL orig8 INSIDE OF (-1000) (-760) 1000 760 FILE [
    fracture_units 500
    device JBX-3030MV
    version 3.1
    field_size 500
    magnify 4
    small_value 0.1
    cd internal 1.4 2.8
    file_name TEST3XXXX.PF
    log_file_name jeol_fracture.log
]
}
```

VSB11 Example

The following illustrates an example VSB11 fracture operation.

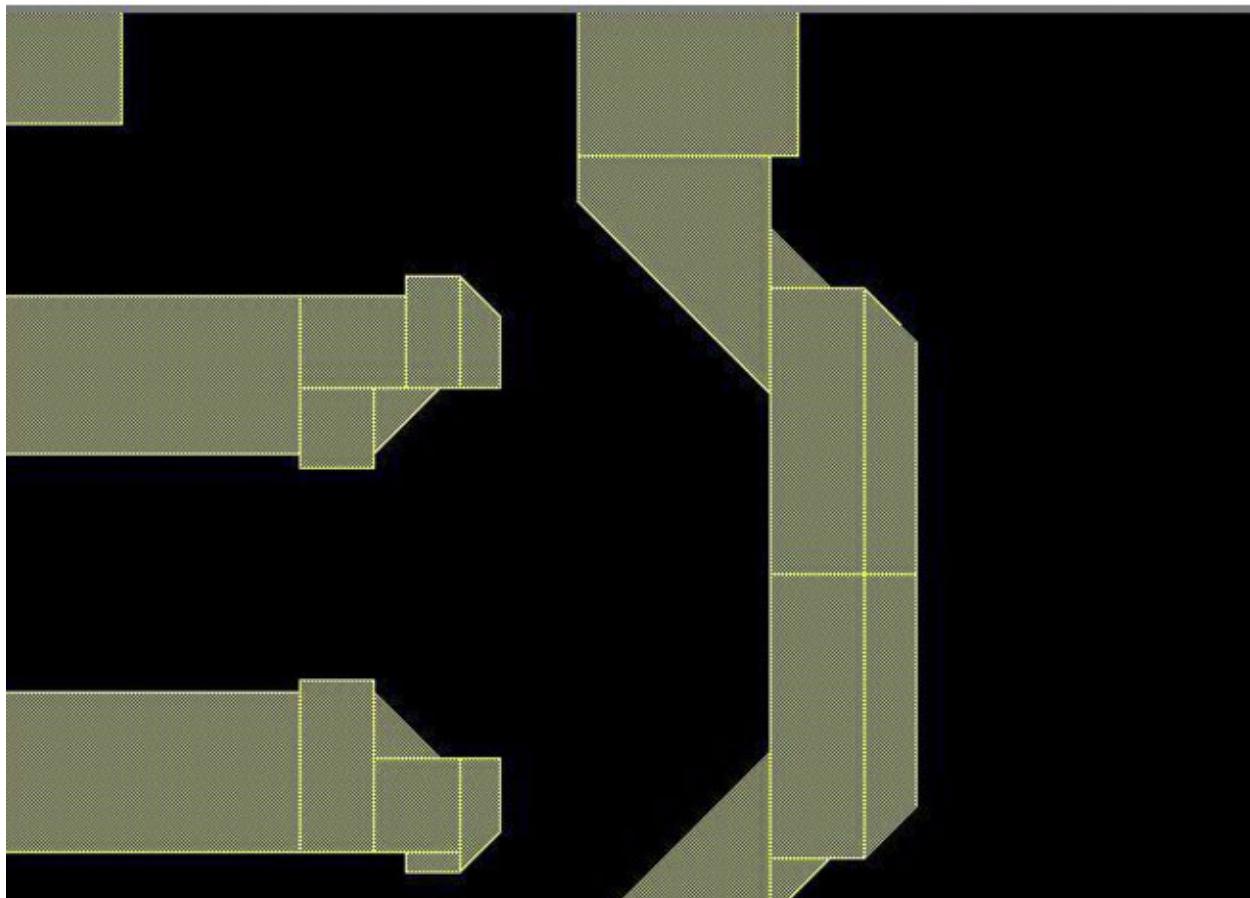
```
///////////////////////////////
// input SVRF file
///////////////////////////////
LAYOUT SYSTEM GDSII
LAYOUT PATH "test.gds"
LAYOUT PRIMARY "*"
PRECISION 1000
RESOLUTION 1
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "test_dummy.gds" GDSII PSEUDO
DRC SUMMARY REPORT "/dev/null"

LAYER poly 4
VSB11_FRACTURE { FRACTURE NUFLARE POLY FILE [
magnify 4
    chip_directory "./chip"
    version 11
    small_value 0.1 //conversion units: 100nm/1.25 = 80
    max_skew_approximation_error 0.01
]
///////////////////////////////
// listing of ./chip directory
///////////////////////////////
total 1808
drwxr-xr-x 2 samplet ae 512 Oct 14 11:29 .
drwxr-xr-x 3 samplet ae 512 Oct 14 11:44 ..
-rw-r--r-- 1 samplet ae 936 Oct 14 11:29 chip.cnf
-rwrxr-xr-x 1 samplet ae 204124 Oct 14 11:28 common.chp
-rwrxr-xr-x 1 samplet ae 22820 Oct 14 11:28 frame.1
-rwrxr-xr-x 1 samplet ae 40724 Oct 14 11:28 frame.2
-rwrxr-xr-x 1 samplet ae 59644 Oct 14 11:28 frame.3
-rwrxr-xr-x 1 samplet ae 96808 Oct 14 11:28 frame.4
-rwrxr-xr-x 1 samplet ae 147600 Oct 14 11:28 frame.5
-rwrxr-xr-x 1 samplet ae 202436 Oct 14 11:28 frame.6
-rwrxr-xr-x 1 samplet ae 86920 Oct 14 11:28 frame.7
-rwrxr-xr-x 1 samplet ae 23400 Oct 14 11:28 frame.8
///////////////////////////////
// content of chip/chip.cnf file
///////////////////////////////
Conversion.System : "Calibre v9.3_4.4 (pre-production) Fri Jun 27
14:10:41 PDT 2003"
Conversion.machine : "jjj"
Conversion.machine.OS : "SunOS"
Conversion.machine.ID : "5.8 Generic_108528-12 sun4u"
Conversion.AddressUnit : "1.25nm"
Chip.Name : "chip"
Frame.Width : 819200
Cell.maxWidth : 102400
Cell.maxHeight : 819200
Cell.subField.size : 51200
Cell.subField.margin : 200
Cell.translate.x : -200
Cell.translate.y : -200
Pattern.arrayCompression : ON
Pattern.Compression : ON
Pattern.Slit.width : 0
Pattern.Set : 1
Common.maxData : 67108864
```

```
Frame.maxData : 67108864
// Common.bdeMaxData : 67108864
// Frame.bdeMaxData : 67108864
// MaxSkewAppoximationError : 8
// Chip.Directory : "./chip"
// small_value : 80
Chip.Size.x : 6382400
Chip.Size.y : 6383232
Chip.Frames : 8
Frame.8.Width : 648832
```

VSB11 Output

Figure 5-1. Fractured Data



HITACHI Example

The following illustrates an example HITACHI fracture operation.

```
///////////////////////////// // input SVRF file //////////////////////////////
LAYOUT SYSTEM GDSII
LAYOUT PATH "GDS/demo_merged.gds"
LAYOUT PRIMARY "demo_wrapper"

DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "GDS/demo_hitachi.gds" GDSII PSEUDO
DRC SUMMARY REPORT "reports/demo_hitachi.rep"
DRC MAXIMUM VERTEX 199

PRECISION 1000
RESOLUTION 10

LAYER POLY 4

LAYOUT MAGNIFY 2.4 // Down 0.6, then up 4.

MY_FRACTURE { FRACTURE HITACHI POLY FILE [
    file_name "demo.pfh3"
    format_type 3
    max_skew_approximation_error 0.1
]
}
```

Micronic Example

The following illustrates an example Micronic fracture operation.

```
///////////////////////////// // input SVRF file //////////////////////////////
LAYOUT SYSTEM GDSII
LAYOUT PATH "GDS/m8051warp_merged.gds"
LAYOUT PRIMARY "m8051warp_wrapper"

DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "GDS/m8051_mic.gds" GDSII PSEUDO
DRC MAXIMUM VERTEX 199

PRECISION 1000
RESOLUTION 10

LAYER POLY 4

LAYOUT MAGNIFY 2.4 // Down 0.6, then up 4.

MY_FRACTURE { FRACTURE MICRONIC POLY FILE [
    version 1.6
    file_name "mdp/m8051.mic"
    log_file_name reports/mic.log
]
}

// bad = MDPVERIFY POLY FILE [
//     input_file mdp/m8051.mic
//     verify_type MICRONIC2DB
// ]

// bad { COPY bad } DRC CHECK MAP bad 300
```

OASIS.MASK Example

The following illustrates an example OASIS.MASK fracture operation.

```
///////////////////////////////
// input SVRF file
///////////////////////////////
LAYOUT SYSTEM GDSII
LAYOUT PATH "$DESIGN_DIR/GDS/hole.gds.Z"
LAYOUT PRIMARY "*"

DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "rules.frac.tst.results.gds" GDSII PSEUDO
DRC SUMMARY REPORT rules.frac.tst.summary.txt
LAYER lay1 1
LAYER lay2 2

hole=lay1 XOR lay2

frac1 { FRACTURE OASIS_MASK hole INSIDE OF EXTENT FILE [
    fracture_units 1000
    magnify 4/1
    file_name TEST1XXXX.OM
    log_file_name TEST1XXXX.OM.log
] }

ver1 = MDPVERIFY hole INSIDE OF EXTENT FILE [
    input_file TEST1XXXX.OM
    magnify 4/1
    verify_type VBOASIS2DB
]
lay1 { COPY lay1 } DRC CHECK MAP lay1 1
lay2 { COPY lay2 } DRC CHECK MAP lay2 2
hole { COPY hole } DRC CHECK MAP hole 3
ver1 { COPY ver1 } DRC CHECK MAP ver1 4
```


Chapter 6

Calibre MDPverify

Calibre MDPverify is a Calibre application that allows you to evaluate fractured MDP data by performing different data comparisons.

The type of comparisons include:

- Fractured data to the original layout data or any Calibre database layer.
- Fractured data in one pattern file to fractured data in a different pattern file.

Calibre MDPverify operates on Calibre layers, data for MEBES, JEOL (including MP records), Hitachi, Micronic, Nuflare, and OASIS-style formats, and job decks. The output is an exclusive flat layer instantiation.

Calibre MDPverify Inputs	189
Calibre MDPverify Outputs	190
MDPVERIFY	191
Region and Shift Default Computation	205
Data Transformation	206
Compare Fractured Data to the Layout Database	207
Compare Fractured Data to Fractured Data	208
Comparing Two VSB Job Decks	215
Using Proper Shift Values	218
Avoiding Problems With Shifting	220
Define the Verification Region	221
Examples	223
Example 1 — MEBES to an Internal Calibre Layer	223
Example 2 — MEBES to MEBES	224
Example 3 — MEBES to JEOL	225

Calibre MDPverify Inputs

Calibre MDPverify inputs are summarized in the following table.

Table 6-1. Inputs to Calibre MDPverify

Comparing Fractured Data to a Layer in the Layout Database	Comparing Fractured Data to Fractured Data
An external file containing the fractured data: MEBES, MEBESJOB, JEOL, Hitachi, Micronic, Nuflare, NUFLARE_MBF, OASIS.MASK, OASIS.MAPPER, or OASIS.MBW	<ul style="list-style-type: none">• An external file containing fractured data (MEBES, JEOL, Hitachi, Micronic, or OASIS), an external directory containing fractured Nuflare (VSB11, VSB12, VSB12i, or NUFLARE_MBF) data, or an external directory containing a Nuflare job deck.• A second external file containing fractured data (MEBES, JEOL, Hitachi, Micronic, OASIS), an external directory containing fractured Nuflare data, or an external directory containing a Nuflare job deck.
A Calibre layer containing original data	A Calibre layer to serve as a placeholder

The fracture pattern files can be one of the supported formats shown in [Table 1-1](#) on page 11.

The MDPVERIFY command can also read compressed MEBES files. Such a compressed file is internally decompressed to the full MEBES file in the *temp_directory*. The decompressed file is removed when Calibre MDPverify is complete. Run time exceptions are possible if the decompression fails or the temporary directory becomes unusable.

Calibre MDPverify Outputs

The output of the Calibre MDPverify operation is a one or more derived layers in the Calibre database. You can use this layer in subsequent operations or write it to the Calibre results database.

Calibre also returns a count of the total number of geometries produced by the Calibre MDPverify operation. This information is written to the transcript. If you include the DRC SUMMARY REPORT statement in your rule deck, it is also written to the summary report as part of the RULECHECK RESULTS STATISTICS.

MDPVERIFY

SVRF mask data preparation command used to verify fractured data.

Usage

```
MDPVERIFY layer1 [...layerN]
[ {INSIDE OF x1 y1 x2 y2 | INSIDE OF LAYER layer_ext | INSIDE OF EXTENT} ]
[FILENAME name_of_file] [MAP mapstring]
FILE{parameter_block_name | '['
    verify_type {JEOL2DB | VSB2DB | MEBES2DB | HITACHI2DB |
        JEOL2JEOL | MEBES2MEBES | VSB2VSB | HITACHI2HITACHI |
        MEBES2VSB | MEBES2JEOL | MEBES2HITACHI | MEBES2MICRONIC |
        MEBESJOB2DB | MEBESJOB2MEBESJOB | MEBESJOB2JEOLJOB |
        MEBESJOB2VSBJOB | MICRONIC2DB | MICRONIC2MICRONIC |
        VSBJOB2VSBJOB | JEOL2VSB | JEOL2HITACHI | JEOLJOB2JEOLJOB |
        VSB2HITACHI | VBOASIS2JEOL | VBOASIS2DB | VBOASIS2VSB |
        VBOASIS2VBOASIS | VBOASIS2MEBES | VBOASIS2MICRONIC}
file1 [file2]
[svrf_layer_name -file1 {layer_name [layer_num [layer_datatype]]}...
 -file2 [{layer_name [layer_num [layer_datatype]]}... | 
 -directIn {layer_name [layer_num [layer_datatype]]}...]]
<SVRFSTART>
    svrf_statement1
    ...
    svrf_statementN
<SVRFEND>
[maximum_output_count limit]
[magnify value1 [value2]]
[rotate rotate1 [rotate2]]
[mirror1 {x | y | x y}]
[mirror2 {x | y | x y}]
[shift {x1 y1 | NONE [x2 y2 | NONE]}]
[reverse_tone {yes | no} [{yes | no}]]
[size size_value]
[mask1 {layout_name | level_number}]
    // applies only to job deck-to-job deck
    // verify_types
[mask2 layout_name]
    // applies only to job deck-to-job deck verify_types
    // and MEBESJOB2VSBJOB
[field_size x [y]]
[field_size_multiplier m]
[input_extent 1 ix1 iy1 ix2 iy2]
[input_extent 2 ix1 iy1 ix2 iy2]
[vboasis_path filepath [-noCheck] [-fastindex [layer datatype]}}
[vboasis_precision_multiplier {n[/d] | AUTO}]
[vboasis_layout_magnify n[/d]]
```

```
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]  
[index_options -topcell -file1 classname [{-file2 | -directIn} classname2]]  
[inject_mask_files]  
[oasis_output_path path]  
[oasis_output_layers {layer [datatype] } | {mapstring layer [datatype] }... }]  
[oasis_output_primary [HDB | DIRECTIN | FILE1 | FILE2 | -explicit topcell_name]]  
[oasis_output_options [-cblock] [-maximum_vertex {mv | ALL}]]  
[runtime_exception_severity [-file_read_error {continue | stop}]  
    [-file_differences {continue | stop}] [-comparison_region {continue | stop}]  
    [-shift_value_overflow {continue | stop}]]  
[direct_FS_access [INPUT] [INTERMEDIATE]]  
[temp_directory directory_path]  
[index_mask_files {ALL | NONE | size_in_MBs}]  
[data_extent {HDB | FILE | DIRECTIN}]  
'  
'  
}
```

Description

The MDPVERIFY command allows you to compare fractured data to pre-fractured data, pattern file to pattern file, OASIS to OASIS, or VSB job deck to VSB job deck. Of the MDP utilities, MDPVERIFY is unique in its ability to read pattern file and job decks, and align pre- and post-fracture data using orientation parameters such as rotation, mirror, shift, and magnify.

Arguments

- *layer1* [...*layerN*]
A required argument supplying the name(s) of an original layer or derived polygon passed to the MDPVERIFY command from the HDB. If you are comparing a pattern file to a database layer, this argument is the name of that layer in the Calibre database. If you are comparing two external files, the layer is still required, but is simply a placeholder layer and can be any layer in the Calibre database.

- {INSIDE OF *x1 y1 x2 y2* | INSIDE OF LAYER *layer_ext* | INSIDE OF EXTENT}
- An optional argument used to define the verification region for the MDPVERIFY operation. This argument must be one of:

- INSIDE OF *x1 y1 x2 y2*

Defines the verification region as a rectangle with the specified coordinates. You must write coordinates relative to the origin of the Calibre database, and enclose negative coordinates in parentheses (). The **x1** and **y1** coordinates specify the lower left corner **x2** and **y2** coordinates specify the upper right corner. For example:

```
INSIDE OF (-123) 123 (-12) 456
```

- INSIDE OF LAYER *layer_ext*

Defines the verification region by supplying a layer whose extent defines the region. *layer_ext* cannot be the same layer as *layer1*.

- INSIDE OF EXTENT

Indicates that the extent of *layer1* defines the verification region. This is the default.

If this argument is not specified, the verification region defaults as follows:

- When comparing an external file to a Calibre layer, the verification region defaults to the equivalent of the INSIDE OF EXTENT keyword.
- If no shift values are specified, the behavior of the MDPVERIFY operation in computing shift and region defaults depend on the type of comparison being made (format to database, format to format). Refer to “[Region and Shift Default Computation](#)” on page 205 for details.
- FILENAME *name_of_file*

An optional method of specifying the name of the first input file for comparisons involving a single file. The *name_of_file* portion of the pathname can contain up to 24 letters, numbers, and symbols. Apostrophes cannot be used.
The FILENAME and input file parameters (input_file parameter) are mutually exclusive. If you use both in the comparison operation, Calibre MDPverify generates an error.
- MAP *mapstring*

An optional keyword that enables multiple concurrent outputs when SVRF is embedded into the MDPVERIFY statement. Each map string must match an output layer in the embedded SVRF, and there must be exactly one map string (and hence concurrent MDPVERIFY SVRF) for each output layer in the embedded SVRF.
- FILE *parameter_block_name*

A required keyword that specifies the output definition parameters contained in a reusable parameter block defined using the [Litho File](#) SVRF statement.
- FILE '[' ... ']'

A required keyword used to specify the output definition parameters. Square brackets ([]) must surround all operation arguments appearing after the FILE keyword. In addition, everything within the square brackets must comply with the following requirements:

 - Keywords and parameters must be on lines strictly between the left and right brackets (that is, they cannot be on the same line).
 - Argument text cannot contain either a left ([) or right bracket (]).
 - You can include comments within the section of the operation set off by square brackets if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.

- When only *file1* is specified, it is the pathname of a data file to be verified against the Calibre layer *layer1*. Alternatively, you can specify this file with the FILENAME parameter.
- When *file1* and *file2* are both specified, they are the pathnames for two files that are verified against each other.
- **verify_type {JEOL2DB | VSB2DB | MEBES2DB | HITACHI2DB | JEOL2JEOL | MEBES2MEBES | VSB2VSB | HITACHI2HITACHI | MEBES2VSB | MEBES2JEOL | MEBES2HITACHI | MEBES2MICRONIC | MEBESJOB2DB | MEBESJOB2MEBESJOB | MEBESJOB2JEOLJOB | MEBESJOB2VSBJOB | MICRONIC2DB | MICRONIC2MICRONIC | VSBJOB2VSBJOB | JEOL2VSB | JEOLJOB2JEOLJOB | JEOL2HITACHI | VSB2HITACHI | VBOASIS2DB | VBOASIS2JEOL | VBOASIS2VSB | VBOASIS2VBOASIS | VBOASIS2MEBES | VBOASIS2MICRONIC | MEBESJOB2MEBESJOB | JEOLJOB2JEOLJOB | VSBJOB2VSBJOB | MEBESJOB2VSBJOB | MEBESJOB2JEOLJOB}**

A required keyword and value pair defining the type of comparison to perform. The input files must match the verify_type. Thus, if you specify MEBES2MEBES, input_file file1 and file2 must both be valid MEBES files.

Note that embedded SVRF does not support job deck to job deck verification. In cases, where the verification extent is not specified with the INSIDE OF keyword, it is set to the extent of the larger input. If inputs to be verified have different extents, they are aligned at the bottom left and verified.

The MEBESJOB2VSBJOB verify_type requires both mask1 and mask2 keywords as well.

Note

 Embedded SVRF does not support *job2job* operations (such as MEBESJOB2VSBJOB, JEOLJOB2JEOLJOB, and so on).

- **input_file *file1* [*file2*]**

A required keyword and value set that supplies the pathname(s) of the file(s) to be verified. The values specified for *file1* and *file2* must be pathnames to the MEBES, JEOL, Hitachi, or Micronic data files, or a Nuflare directory.

- **svrf_layer_name -file1 {layer_name [layer_num [layer_datatype]]}... [-file2 {layer_name [layer_num [layer_datatype]]}... | -directIn {layer_name [layer_num [layer_datatype]]}...]**

An optional keyword used to map input source layers to SVRF layer names.

There are several common usage forms of this syntax, described as follows:

- **svrf_layer_name -file1 *layer_name* [-file2 *layer_name*]**

This form provides SVRF layer names with MDPVERIFY file inputs. In this form, -file1 specifies a layer from an input pattern file, and -file2 specifies a layer from an input pattern file or database. When using -file1 and -file2, only one layer can be specified except for the OASIS.MASK format. The first form provides SVRF layer

names with MDPVERIFY file inputs. You can map the input file order (file1, file2) to the MDPVERIFY comparison type:

- *patternfile1*2DB: For instance, MEBES2DB

In this case, the MEBES pattern file is again “file1,” and the database is “file2.”

- *patternfile1*2*patternfile2*: For instance, MEBES2JEOL

In this case, the input MEBES pattern file is “file1” and the -file1 *layer_name* comes from that file. The JEOL input pattern file is “file2,” and the -file2 *layer_name* originates from that file.

- svrf_layer_name -file1 {*layer_name* [*layer_num* [*layer_datatype*]])... [-file2 {*layer_name* [*layer_num* [*layer_datatype*]])...]

This form is an expanded version of the previous form that allows you to specify more than one layer. However, when specifying more than one layer, the *layer_num* is required. This form optionally allows you to also specify layer datatypes. For the MEBESJOB input type, *layer_num* must be set to the job deck level to be verified.

- svrf_layer_name -file1 {*layer_name* [*layer_num* [*layer_datatype*]])...[-directIn {*layer_name* [*layer_num* [*layer_datatype*]])...]

In this form, the syntax follows the previous form except -file2 is now substituted by -directIn, which means that you can use a direct-input database instead (OASIS).

When using a direct-input database, you can specify pairs of SVRF layer names and source layer numbers with the -directIn option.

- <SVRFSTART>

An optional keyword indicating the beginning of the embedded SVRF statement group.

- *svrf_statement1* [... *svrf_statementN*]

An optional keyword consisting of an SVRF statement. Each SVRF statement must be on a line of its own. Calibre truncates any SVRF line to 1023 characters if you exceed this value.

You must specify a single rule check within the embedded SVRF statement group. Refer to the *Calibre Verification User’s Manual* for more information about rule checks.

You cannot specify left- or right-brackets within the embedded SVRF statement group. Therefore, any occurrence of brackets must be replaced as follows:

Left bracket ([)	<LB>
Right bracket (])	<RB>

The PRECISION setting of the embedded SVRF statements must match that of the Calibre rule deck containing the MDPVERIFY statement.

- <SVRFEND>

An optional keyword indicating the end of the embedded SVRF statement group.

Note

 The left and right angle brackets (< >) for <SVRFSTART>, <SVRFEND>, <LB>, and <RB> are literal and required.

- `maximum_output_count limit`

An optional keyword and value pair defining the maximum number of output polygons to be generated. This is useful for early detection of errors that cause a large number of outputs to be generated, such as misalignment. Specifying “all” means there is no limit. The value is a lower bound to the number of outputs that is generated. When the output count exceeds this value, the verification process halts, but any pending jobs on threads or remote processes still completes and stores their output.

The default is 2000000.

- `rotate rotate1 [rotate2]`

An optional keyword and value pair instructing MDPVERIFY to apply an inverse rotation to the file(s) before performing the comparison. Allowed values are 90, 180, and 270.

Rotation is in the clockwise direction (the inverse of the FRACTURE rotation, which is counter-clockwise.) When using rotate to undo a rotation that was performed when generating the fractured file, the rotate values must match the rotate values used in the FRACTURE command.

- When one value is specified, the rotation is applied to the external file.
- When two values are specified, the first is applied to the first input file and the second to the second input file.

Note

 The intent behind Calibre MDPverify is to reverse the sequence of transformations applied when fracturing. Because of this, when mirror, rotate, or magnify keywords are used in the same MDPVERIFY operation, the order of application (magnify - rotate - mirror) is used, regardless of their order in the rule file. This is the reverse of the order used when fracturing.

The default is 0 0.

- `magnify value1 [value2]`

An optional keyword and value pair instructing MDPVERIFY to apply an inverse magnification to the file(s) before performing the comparison. The values can be any double floating point numbers.

This option magnifies the first external file by (1/*value1*), and if a second value is specified, magnifies the second external file by (1/*value2*).

Default values are 1 1.

Alternatively, you can specify the magnify value in fractional form using a forward slash (/) between the numerator and denominator

Typically, magnify is used when magnification was performed when generating the fractured file, and must be un-done before verification. The magnify values must match the magnify values used in the fracture command. Thus, if you used magnify 4 (to match the mask/wafer scale) when fracturing, you would specify magnify 4 here, which causes MDPVERIFY to magnify by 0.25.

- mirror1 {*x* | *y* | *x y*}

An optional keyword and value pair instructing MDPVERIFY to apply an inverse mirror to the file1 before performing the comparison. Mirroring is across the axis running through the center of the region.

When both mirror and rotate keywords are used in the same MDPVERIFY operation, mirror is always applied after rotation, regardless of their order in the rule file.

The default is no mirror.

- mirror2 {*x* | *y* | *x y*}

An optional keyword and value pair instructing MDPVERIFY to apply an inverse mirror to the file2 before performing the comparison.

When both mirror and rotate keywords are used in the same MDPVERIFY operation, mirror is always applied after rotation, regardless of their order in the rule file.

The default is no mirror.

Refer to the section “[Region and Shift Default Computation](#)” on page 205 for details.

- shift {*x1 y1* | NONE [*x2 y2* | NONE]}

An optional keyword that specifies coordinates to align input data for comparison. The coordinates specify the location of the lower-left corner of the corresponding file extent (not fracture extent) in the Calibre coordinate system (after all inverse transformation is completed) before comparison.

For example, for a fracture file to fracture file comparison (such as verify_type JEOL2HITACHI), it is common to use the lower-left of the fracture region.

If this keyword is not specified and the comparison is between a fracture file and the hierarchical database (HDB), the correct value is calculated automatically by assuming the comparison region is the same as the original fracture region.

There are multiple sources of input data (an HDB, a fracture file, or a layout file). Each source data is moved to a common alignment for comparison. There are two components of each data movement:

- Justification of the pattern in some cases (data is moved such that the lower-left boundary is at the lower-left of the verification region).
- The shift value (default is 0 0).

It is common for pattern file applications not to specify a shift but to have justification occur. Specifying “shift NONE” prevents both justification and shift.

Consider three common applications:

- pattern-to-DB comparison (for example, verify_type JEOL2DB)

The verification region is specified in the coordinate system of the HDB (which does not move). When the boundary size is the same, the pattern is justified to align to the HDB. The default shift (0 0) is usually used. Alternatively, you can use:

- shift *x1 y1*: Use if you have a need for additional shift, but this is typically unnecessary.
- shift NONE: If you need to deactivate all justification and shift.
- pattern-to-pattern comparison

Usually, both patterns have lower-left origins, and no additional justification and shift is required. The most common usage (in order):

- No shift keyword at all (equivalent to shift 0 0 0 0)
- shift *x1 y1 x2 y2*
- shift NONE *x2 y2*
- shift *x1 y1* NONE
- OASIS-to-OASIS (dual-direct-data-entry (DDE))

In this case, the use model is to compare two OASIS layouts using “verify_type VBOASIS2VBOASIS” and “shift NONE NONE”. Subsequently, there is no need to move any of the input data.

The use model for this scenario is to compare two OASIS layouts using “verify_type VBOASIS2VBOASIS.” There is no need to move any of the input data and therefore you must assert that the OASIS.MASK data should not shift; “shift NONE shift NONE” should be used for this case.

In other words, for VBOASIS2DB, “shift NONE [shift NONE]” is required since a shift is always applied to VBOASIS files. The only way to turn the shift off and to preserve input coordinate location for OASIS DDE (VBOASIS) files is to set shift to NONE.

Note

 For verify_types with job deck inputs, transformation or reverse tone parameters are not allowed; use of “shift” is permitted.

- reverse_tone {yes | no} [{yes | no}]

An optional keyword and value pair that inverts one or both inputs before comparison. Tone reversal may not be used with embedded SVRF.

The default is “no no”.

Do not use the reverse_tone option with embedded SVRF, since some one-layer SVRFs require unmerged data. Instead, use the following embedded SVRF sequence:

- a. Pass in an extent layer as one of the inputs.
- b. Take the layer that is read in from the pattern file.
- c. Use a Boolean NOT operation to reverse tone the layer and use it for comparison.

For example:

```
verify = MDPVERIFY input1 bulk
    INSIDE OF EXTENT
    FILE      [
        input_file           fracture.jeol
        svrf_layer_name     fracture.jeol
        verify_type          jeol2db
        <SVRFSTART>
            rev_tone = bulk NOT fracture.jeol
            output { rev_tone XOR input1 }
        <SVRFEND>
    ]
```

- size *size_value*

An optional keyword and coordinates pair instructing MDPVERIFY to apply a “size underover” operation to the output polygons as a final step in the processing. The *size_value* must be a non-negative value, expressed in user units in the Calibre coordinate system.

Note

 This operation is useful in suppressing small snapping errors that may be created due to the grid size mismatch during the fracturing operation.

The default is 0, which means no sizing is performed.

Note

 The size keyword does not take effect if embedded SVRF is used (the <SVRFSTART> ... <SVRFEND> block). If you want to size the MDPVERIFY XOR results, you can place the size command as part of the embedded SVRF block, or use an alternate SVRF command (for example, the SVRF command “size poly by 0.005 under over” is equivalent to a “size 0.005” statement within MDPVERIFY).

Figure 6-1. SIZE Keyword Alternatives for Embedded SVRF

```
MDPVerify_VSB11_ESVRF = MDPVERIFY POLY INSIDE OF EXTENT FILE [
    input_file           ./input/FRACTXinp.VSB11
    verify_type          vsb2db
    svrf_layer_name pattern1
    //size 0.00125
    // declaring size with embedded svrf generates an error
    <SVRFSTART>
        output_tmp = POLY XOR pattern1

        //alternatively, use size in the embedded SVRF block
        output { size output_tmp by 0.001 underover }
    <SVRFEND>
]

...
//another method of using size
sized_ver { size MDPVerify_VSB11_ESVRF by 0.001 underover } DRC CHECK
MAP \
    sized_ver 1111
```

- **mask1 {*layout_name* | *level_number*}**

Applies only to job deck-to-job deck verify_types (such as VSBJOB2VSBJOB, MEBESJOB2MEBESJOB, JEOLJOB, and MEBESJOB2VSBJOB). An optional keyword and value pair specifying the names of the layouts (or the mask1 level number for MEBESJOB or JEOLJOB) in the job deck to be verified. This keyword can be omitted if only one layout exists in the first input VSB job deck. This keyword is required for MEBESJOB2VSBJOB along with mask2.

- **mask2 *layout_name***

Applies only to job deck-to-job deck verify_types (such as VSBJOB2VSBJOB, MEBESJOB2MEBESJOB, JEOLJOB, and MEBESJOB2VSBJOB). An optional keyword that specifies the name of the second job deck if the VSBJOB, MEBESJOB, or JEOLJOB. This keyword can be omitted if only one layout exists in the second input VSB job deck. This keyword is required for MEBESJOB2VSBJOB along with mask2.

- **field_size *x* [*y*]**

Specifies the nominal size of the flat verification section in user units in the HDB coordinate system. The effective real section size is calculated from the following formula:

$(\text{field_size_multiplier} * \text{field_size}) / \text{magnify}$, where magnify is *n1* [/ *d1*].

If only *x* is specified, then the section is a square with side length *x*. Otherwise, the section is a rectangle with width *x* and height *y*. The default setting is 1000 500.

Note

 For input data with a 1X magnification, the field_size_multiplier should be set to 1 to achieve the desired section size.

- `field_size_multiplier m`

An optional keyword that specifies the multiplier in calculating the effective real section size. The actual formula can be found in the `field_size` section. The default setting is 4.

- `input_extent 1 ix1 iy1 ix2 iy2 input_extent 2 ix1 iy1 ix2 iy2`

An optional keyword that is used to override the extent for an input data source. Typically, the input data sources provide an explicit definition of the extent of the corresponding layout. For instance, each fracture format contains this information in the file header. However, files of some formats may not contain such information, in which case the default layout extent is the extent of geometry in the file.

The input source extent is important because it plays a large role in the inference of the transform required to align files. Often the extent of geometry in a file is not the desired extent of the mask part. Use this keyword to specify the desired extent, in user units of the input source.

This keyword applies only to inputs of type VBOASIS and VSBJOB.

- `vboasis_path filepath [-noCheck] [-fastindex [layer datatype]]`

An optional keyword that specifies the path of a single OASIS DDE file. This allows you to bypass the HDB and instead take input directly from the OASIS DDE file.

The `-noCheck` option can be specified when checks on the input OASIS DDE layout file are not possible because it has not yet been generated. This would typically happen when the file is expected to be generated by another SVRF command and thus is not available during parsing for checking purpose. When using `-noCheck`, care should be given to specifying the file path and associated PRECISION values to avoid errors that are encountered much later.

The `-fastIndex` keyword enables a “fast mode” of index generation if the input file is an OASIS.MASK file. The file created in fast mode is valid only if the input OASIS.MASK file contains a single layer-datatype pair. Fast mode index generation should be avoided when there are multiple layers or datatypes in the input. By default, fast mode index generation assumes layer-datatype of 0-0. You can override these values by specifying layer-datatype pair as an argument to `-fastIndex` option.

Compressed layouts (those ending in the extension .gz) are not supported by `vboasis_path`.

- `vboasis_precision_multiplier {n/d} | AUTO`

An optional keyword used to multiply the PRECISION value by the supplied number *n*, and then scale the data from the file by the same amount to retain the same absolute scale as the original PRECISION setting. The default value for *n* is 1. The */d* is used to express a rational number as a fraction (for example, a multiplier of 1.5x would be declared as “3/2”).

There may be occasions when you want to re-interpret the PRECISION setting of the OASIS DDE file as specified using the `vboasis_path` keyword. For instance, you may want a larger PRECISION setting to allow finer shape modifications in embedded SVRF code.

If AUTO is supplied instead of a ratio, Calibre attempts to infer the correct ratio but fails if it is not a rational number, or if the combination of inferred numerator and denominator would cause arithmetic overflow. This keyword requires the use of embedded SVRF.

- **vboasis_layout_magnify *n[/d]***

An optional keyword that aligns the OASIS-direct-input coordinate system with the HDB coordinate system by adjusting its scale. The scale is specified as the numerator and denominator of a rational number.

This keyword is primarily used to apply embedded SVRF at the scale of the mask, but after all inverse transforms have been applied to align the MDP file inputs with the HDB coordinate system. In this case one would scale up the HDB coordinate system using LAYOUT MAGNIFY, and also use this keyword to apply the same scale to the direct-input database.

- **vboasis_injection [-size *bin_size* | -size_factor *size_multiplier*] [-preserve 0 | 1]**

An optional keyword that specifies the size of the cells that should be binned and the size of each bin (*bin_size*). This option is used when an OASIS DDE file (specified using vboasis_path) contains cells that have large amounts of data and a large extent. Partitioning large cells into bins prevents performance degradation during section mode processing.

You can control the bin size by providing a *size_multiplier* using the option -size_factor *size_multiplier* (the default value is 1.0).

Alternately the *bin_size* can be explicitly specified using the option -size *bin_size* in microns. Any cell whose extent has width or height more than the *bin_size* is considered for binning.

The vboasis_injection option creates temporary data which may be reused if it is preserved using the -preserve option (the default is 0).

The temporary data is read or written in a directory named *file_path.lcb* where *file_path* is argument specified to vboasis_path. The location of this directory is same as that of *file_path*. This behavior can be changed by specifying a colon-separated list of directories using the UNIX environment variable MDPIIndexSearchPath. The directories are searched in the order they are listed for reading or writing the temporary directory.

- **index_options -topcell -file1 *cellname* [{-file2 | -directIn} *cellname2*]**

An optional keyword used to apply OASIS input module and indexing options. The -topcell option specifies the top cell name for indexing. This is identical to the oasisIndex [*topcellname*] utility in Calibre MDPview.

This parameter is used in cases where you have OASIS files with more than one top cell. The oasisIndex and the indexing capability of Calibre FRACTURE, Calibre MDPverify, MDP EMBED, and DENSITY CONVOLVE all assume that there is only one top cell. If there is more than one top cell, then oasisIndex and the other batch direct-data-entry tools by default pick the top cell that has the deepest hierarchy. If you want to pick a top cell that has a shallow hierarchy, you can instead specify the top cell name explicitly with -topcell.

The -file 2 keyword indicates a second input. The -directIn allows you to use a direct-input database (OASIS).

- `inject_mask_files`

An optional keyword that applies OASIS injection to any OASIS or OASIS-derivative file specified by the `input_file` keyword. By default, injection does not apply to these files. Injection parameters are taken from the existing `vboasis_injection` keyword.

- `oasis_output_path path`

An optional keyword that diverts output geometry from the HDB to a new OASIS file located at *path*. The path should be on a high-performance file system. The OASIS file has a two-level hierarchy in which the sections form uniquely placed cells. The PRECISION setting of the OASIS file matches the PRECISION setting of the HDB. You can optionally use environment variables to specify the path as well.

- `oasis_output_layers {layer [datatype]} | {mapstring layer [datatype]}...}`

An optional keyword that specifies the OASIS layer and datatype numbers for SVRF layers written to an OASIS file. This keyword requires specification of the `oasis_output_path` keyword. The first form works if the embedded rule deck produces only a single layer. The second form works for multiple layers, and requires the use of the MAP secondary keyword in the MDPVERIFY SVRF calls. The map string in the SVRF call must match the *mapstring* value in this keyword specification. All output layers must be mapped using this keyword.

When `oasis_output_layers` is specified in section-based processing, additional information appears in the transcript, reporting the DDO per-rule output shape counts.

```
DIRECT OUTPUT RuleCheck flat1::<1> COMPLETED. Number of Results =
3375 (3375)
DIRECT OUTPUT RuleCheck flat2::<1> COMPLETED. Number of Results =
3375 (3375)
```

- `oasis_output_primary [HDB | DIRECTIN | FILE1 | FILE2 | -explicit topcell_name]`

An optional keyword that specifies the source of top cell name in OASIS output. When specified as:

HDB — MDPVERIFY uses the topcell name from the HDB as the topcell name of OASIS output.

DIRECTIN — MDPVERIFY uses the topcell name from the `vboasis_path` input.

FILE1 — MDPVERIFY uses the topcell name from the first `input_file` (only allowed for MDPVERIFY with VBOASIS2*).

FILE2 — MDPVERIFY uses the topcell name from the second `input_file` (only allowed for MDPVERIFY with *2VBOASIS).

-explicit *topcell_name* — MDPVERIFY uses the *topcell_name*.

The default setting for `oasis_output_primary` is “-explicit top”.

- `oasis_output_options [-cblock] [-maximum_vertex {mv | ALL}]`

An optional keyword used to specify OASIS output options. With the “-cblock” option, the OASIS output is compressed using CBLOCK record. The use of cblock can reduce the output file size.

The maximum number of vertices (*mv*) in output polygons can be limited using the -maximum_vertex parameter. The default is 8192 and the symbolic value ALL corresponds to the maximum setting 4294967295.

- `runtime_exception_severity [-file_read_error {continue | stop}] [-file_differences {continue | stop}] [-comparison_region {continue | stop}] [-shift_value_overflow {continue | stop}]`

An optional parameter that specifies an exception type and how Calibre proceeds when the exception is encountered during the run. The exception types include:

- `file_read_error` (MDP exit status: 20) — MDPVERIFY encounters an input file that cannot be read. The default value is stop.
- `file_differences` (MDP exit status: 21) — MDPVERIFY encounters files of the same format that have different units, extents or transformations. The default value is stop.
- `comparison_region` (MDP exit status: 22) — The MDPVERIFY comparison region and chip extents are possibly incorrect. The default value is stop.
- `shift_value_overflow` (MDP exit status: 23) — The MDPVERIFY shift values are in overflow. The default value is stop.

When “continue” is specified for any of these options, Calibre attempts to finish operations and executes any remaining statements in the rule file, even though MDPVERIFY results may not be valid. A warning message is generated in the end.

When “stop” is specified, the MDPVERIFY operation immediately exits the Calibre run, and an error message is generated.

For VSBJOB2VSBJOB, no transformation and reverse tone are allowed (though shifting is allowed). For VSBJOB2VSBJOB, the two input job decks must have same mask size; all chips in job decks must have the same chip units.

- `direct_FS_access [INPUT] [INTERMEDIATE]`

An optional keyword that permits remote processes in a Calibre MTFlex invocation to directly access a file system.

`INPUT` — Causes remote processes to directly access the argument to `vboasis_path` (including adjunct data such as injection files) and any VSB12 and OASIS files that are arguments to `input_file`.

`INTERMEDIATE` — Causes direct writing of temporary intermediate files relating to use of the `oasis_output_path` keyword.

- `temp_directory`

An optional keyword that specifies a general-purpose temporary directory. This is currently used to temporarily store decompressed input MEBES files. The path should be on a high-capacity disk device as the resulting temporary files are accessed frequently.

- `index_mask_files {ALL | NONE | size_in_MB}`

An optional keyword used to control the generation of an index file. For several formats, a persistent additional index file associated with the input layout is generated. In general, the index file is helpful in maintaining good performance, but you can control whether or not to generate a file on disk using `index_mask_files`. This option is supported for MEBES and OASIS.MASK files.

`ALL`— Generates index files for any pattern file. This is the default setting.

`NONE`— Prevents the creation of an index file on disk (an “index” module is still created in the program without performance degradation).

`size_in_MB`— Specifies the threshold size in MBs for the pattern file. For any pattern file that is greater in size than `size_in_MB`, an index file is generated.

- `data_extent {HDB | FILE | DIRECTIN}`

An optional keyword that specifies which layout extent to use for the operation. This keyword is applicable only when the `vboasis_path` option is specified. If `data_extent` is set to `FILE` or `DIRECTIN`, the extent from the direct input is used; otherwise, the extent from the hierarchical database is used. Use of `data_extent` is not allowed with `INSIDE OF LAYER` or `INSIDE OF EXTENT` or `INSIDE OF x1 y1 x2 y2`. The default value is `HDB`.

Region and Shift Default Computation

When the shift option is not specified, the behavior of Calibre MDPverify depends on the type of comparison being performed:

- **Format-to-DB comparison** (for example, `MEBES2DB`): If the shift option is not specified, coordinates are computed automatically using the verification region. This is possible because the verification region is always available in this mode.
- **Format-to-format comparison** (for example, `MEBES2JEOL`): If the file extents AND all transform parameters are the same, the verification region is computed automatically as the common inverse extent (for example, the extent after transformation). The default shift values for format-for-format comparison is always (0,0), regardless of whether the verification region is specified.

Data Transformation

The MDPVERIFY operation provides transformation controls to ensure that the data you are comparing is the same scale, same orientation, and has the same relative origin.

The transformation functionality for MDPVERIFY actually performs reverse transformations. You enter the transformation that was performed on the original layout data when the pattern file was first generated, and MDPVERIFY reverses the transformation. To accomplish this, the individual transformations operate in reverse and the sequence in which the transformations are applied is reversed. For example, when fracturing, the fracture region is isolated, mirror is applied, and then rotation. Continuing with the example, when comparing data with MDPVERIFY, rotation is applied first, then mirror, and then the verification region is isolated.

Table 6-2. Effect of Transformation in FRACTURE and MDPVERIFY

Fracture Transform	Effect	MDPVERIFY Transform	Effect
MAGNIFY 4	magnify by 4	MAGNIFY 4	magnify by .25
ROTATE 90 ¹	rotate 90 degrees counter clockwise	ROTATE 90 ²	rotate 90 degrees clockwise
MIRROR x y	mirror across x then y	MIRROR1 x y MIRROR2 x y	mirror across y then x
REVERSE_TONE	reverses tone	REVERSE_TONE yes	reverses tone

1. If both mirror and rotate are specified during fracturing, mirror is always performed before rotation.
2. If both mirror and rotate are specified for MDPVERIFY, rotation is always performed before mirror.

Transformation of data to reflect the requirements of the different fracture formats occurs automatically, with no user input. This includes:

- Adjusting the data so that it is relative to the same origin.
- Adjusting the data to the same PRECISION setting.

For example, the following assumes that the verification region and all transform parameters for all file inputs are known, whether by user specification or default specification. For each file input:

1. Data for each format is converted to a common (Cartesian) coordinate system.
2. Magnification and unit conversion are inverted simultaneously.
3. Rotation is inverted.
4. Mirror is inverted.
5. Data is shifted so that the lower-left extent corner lies at the point specified by the “shift” keyword.

6. Then XOR or embedded SVRF is executed on the results of step 5 within the verification region.

Note

 Precision refers to the smallest allowed shape. In MEBES pattern files, the smallest allowed shape is defined by the address_size value, which is written in microns. In JEOL pattern files, the smallest allowed shape is defined by the fracture_units value, which is expressed as 1/grid size in microns. For layout design data, the smallest allowed shape is defined by the internal database precision (not the layout grid). You define the database precision using the [Precision](#) statement which is set to 1/DBU.

The precision you define for the Calibre run using the SVRF [Precision](#) statement defines the precision used for all data read into to the Calibre database. This includes the external pattern files. Thus, while the unit size controls any off-grid snapping while fracturing data to create a pattern file, the Precision defines the off-grid snapping for data being evaluated using MDPVERIFY.

Note

 You must take precision into account when considering magnification. For example, assume you have a pattern file with a unit size of 5nm (.005 microns) that was generated using a magnification of 4. Before comparing the fractured data to either the original layout or another file, you use the magnify 4 option to perform an inverse magnification (magnify by .25) which results in the smallest feature size shrinking from 5nm to 1.25 nm (.00125). In order to express the coordinates without snapping, you must use a precision of 4000, which equates to the smallest allowed figure being greater than .00025. A precision of 800 is the lowest acceptable value in this example.

Compare Fractured Data to the Layout Database	207
Compare Fractured Data to Fractured Data	208
Comparing Two VSB Job Decks	215
Using Proper Shift Values.....	218
Avoiding Problems With Shifting	220
Define the Verification Region	221

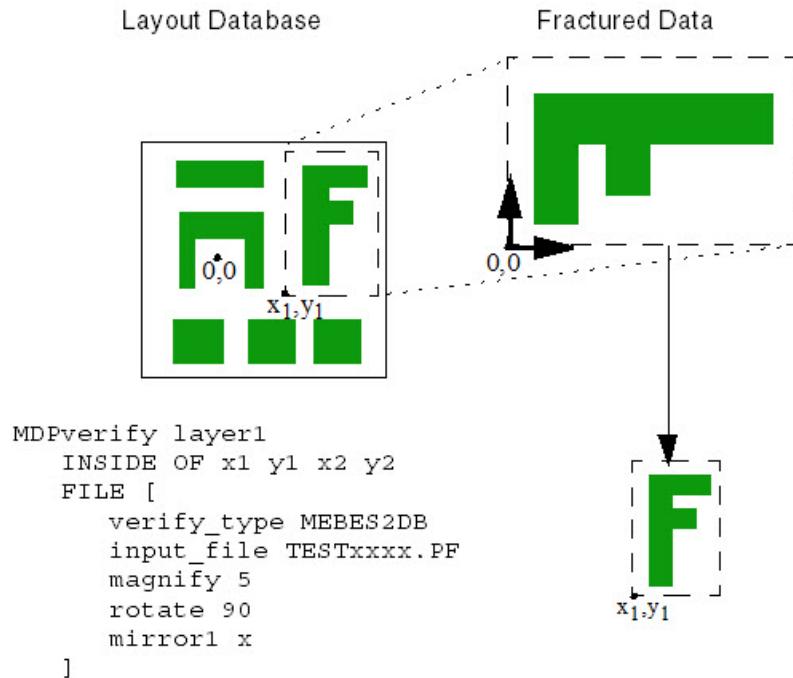
Compare Fractured Data to the Layout Database

After fracturing, the fractured data can differ from the original data using a number of different methods.

The methods include the following:

- The origin can be shifted to meet fracture format requirements.
- The unit size can be different than the database grid.

- Data can be mirrored, rotated, or scaled.
- Tone can be reversed.

Figure 6-2. Comparing Fractured Data to the Layout Database

By configuring the MDPVERIFY operation using the same transformation settings that you used to fracture the data originally, you compensate for these differences. After transformation and conversion, the verification region and the data in the region is equivalent.

Compare Fractured Data to Fractured Data

If you know the transformations that were applied when generating the fractured files from the original layout for both files, you can use these transformations as input to the MDPVERIFY operation.

Since MDPVERIFY allows you to perform different transformations on each of the input files, you can bring both files into alignment with the original layout by specifying the transformations that were applied when the data was fractured.

Note that for MDPVERIFY pattern file comparisons (for example, VSB2VSB, MEBES2VSB), the pattern files must use the same units for fracture_units and address_size, otherwise MDPVERIFY generates an error.

Figure 6-3. Fracture Transformations for Two Sets of Data

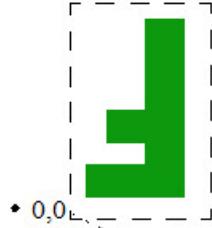
MEBES Fracture

```
mebes_2 { fracture mebes Layer1
inside of x1 y1 x2 y2 file [
mode 5
file_name TEST2xxxx.PF
address_size 0.002
magnify 4
rotate 180
]
}
```

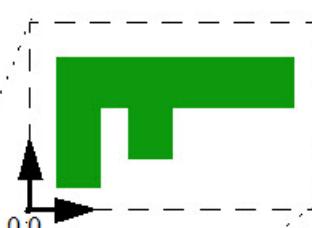
JEOL Fracture

```
jeol_1 { fracture jeol Layer1
inside of x1 y1 x2 y2 file [
fracture_units 500
device JBX-3030MV
version 3.1
field_size 500
file_name
TEST1xxxx.JEOL
magnify 5
rotate 90
mirror x
]
}
```

MEBES Data

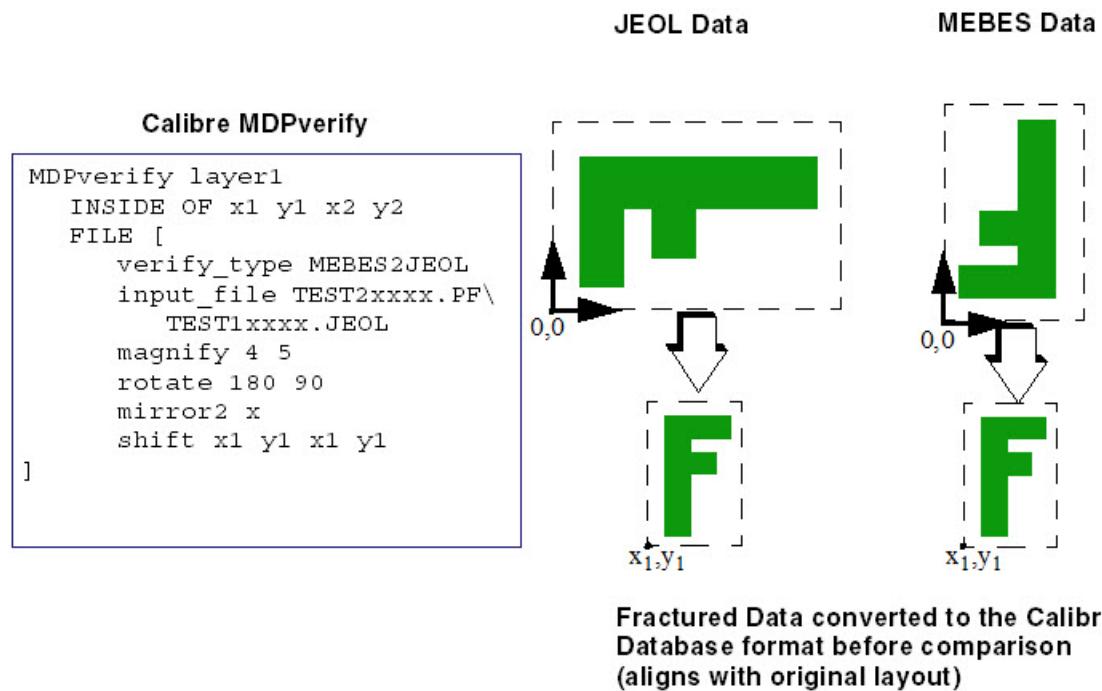


JEOL Data



Original Layout Database

Figure 6-4. Calibre MDPverify Transformations for Two Sets of Data



If you do not know the transformations that were applied and the coordinates of the fracture region, you must calculate the transformations necessary to bring the two sets of data into alignment with each other.

Figure 6-5. Transforming Only One Set of Fractured Data

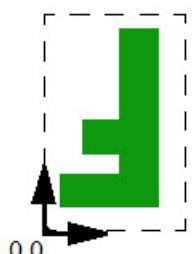
MEBES Fracture 1

```
mebes_1 { fracture mebes Layer1
inside of x1 y1 x2 y2 file [
    mode 5
    file_name TEST1xxxx.PF
    address_size 0.025
    magnify 1
    rotate 0
]
}
```

MEBES Fracture 2

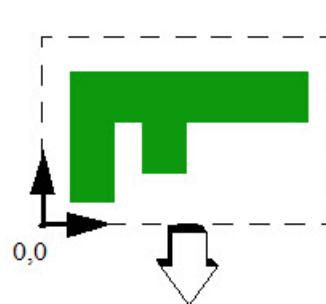
```
mebes_2{ fracture mebes Layer1
inside of x1 y1 x2 y2 file [
    mode 5
    file_name TEST2xxxx.PF
    address_size 0.025
    magnify 2
    rotate 90
    mirror x
]
```

Fractured Data 1



Calibre MDPverify

Fractured Data 2



Fractured Data converted to the Calibre Database format before comparison

```
MDPverify layer1
INSIDE OF x1 y1 x2 y2
FILE [
    verify_type MEBES2MEBES
    input_file TEST1xxxx.PF\
        TEST2xxxx.PF
    magnify 1 2
    rotate 0 90
    mirror2 x
]
```

Caution

-  When comparing one fracture format database to another, there are always three different coordinate domains involved: the SVRF coordinate domain, and the coordinate domains in the two fractured databases for comparison.

It is important to remember that the operations in MDPVERIFY are based on the **SVRF coordinate domain**. Because of this, in MDPVERIFY, you must set the shift value to the coordinates of the bottom-left corner of the contents in the pre-fractured database domain.

Example 1: Comparing Two Databases With the Same Extent

In this example, two databases of different FRACTURE formats (MEBES and HITACHI) with the same extent are compared.

The initial FRACTURE code is as follows:

For MEBES Fracture:

```
mebes_5 {
    fracture mebes Layer1 inside of -1 3 100 138 file [
        mode 5
        file_name mebesmod5.pf
        log_file_name mebesmod5.log
        address_size 0.025
        magnify 2
        mirror x y
    ]
}
```

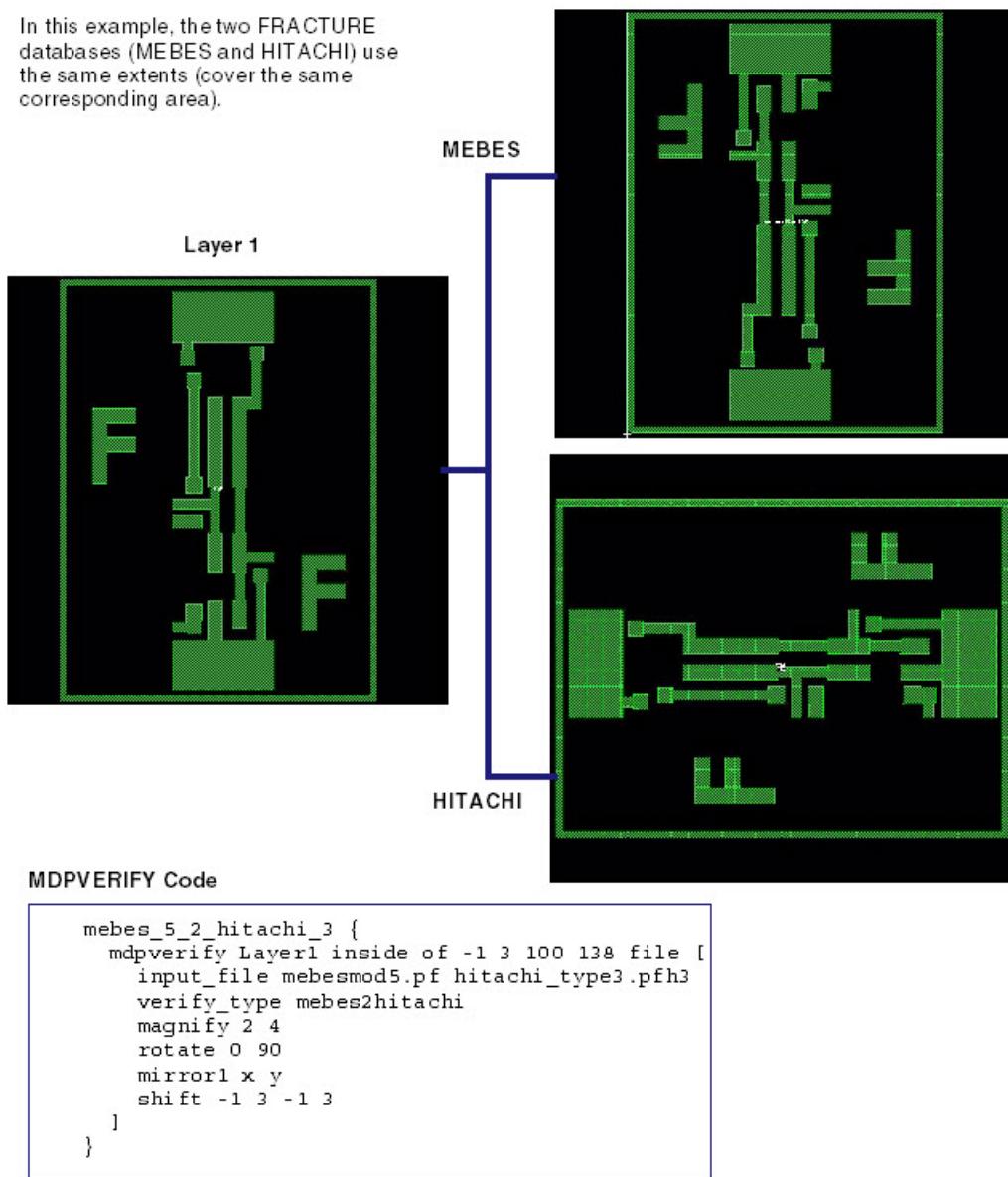
For the HITACHI Fracture:

```
hitachi_3 {
    fracture hitachi Layer1 inside of -1 3 100 138 file [
        format_type 3
        file_name hitachi_type3.pfh3
        log_file_name hitachi_type3.log
        max_skew_approximation_error 0.010
        magnify 4
        rotate 90
    ]
} drc check map hitachi_3 oasis 0 0
```

The results of these fracture operations are shown in [Figure 6-6](#).

Figure 6-6. Comparing Two Databases With the Same Extent

In this example, the two FRACTURE databases (MEBES and HITACHI) use the same extents (cover the same corresponding area).



Example 2: Comparing Two Databases With Different Extents

In this example, two databases of different FRACTURE formats (JEOL and VSB12) with different extents are compared.

The initial FRACTURE code is as follows:

For the JEOL Fracture:

```
jeol_30 {
    fracture jeol Layer1 inside of -1 3 100 138 file [
        version 3.0
        file_name jeol_30.jeol
        log_file_name jeol_30.log
        fracture_units 1000
        magnify 3
        mirror x
    ]
}
```

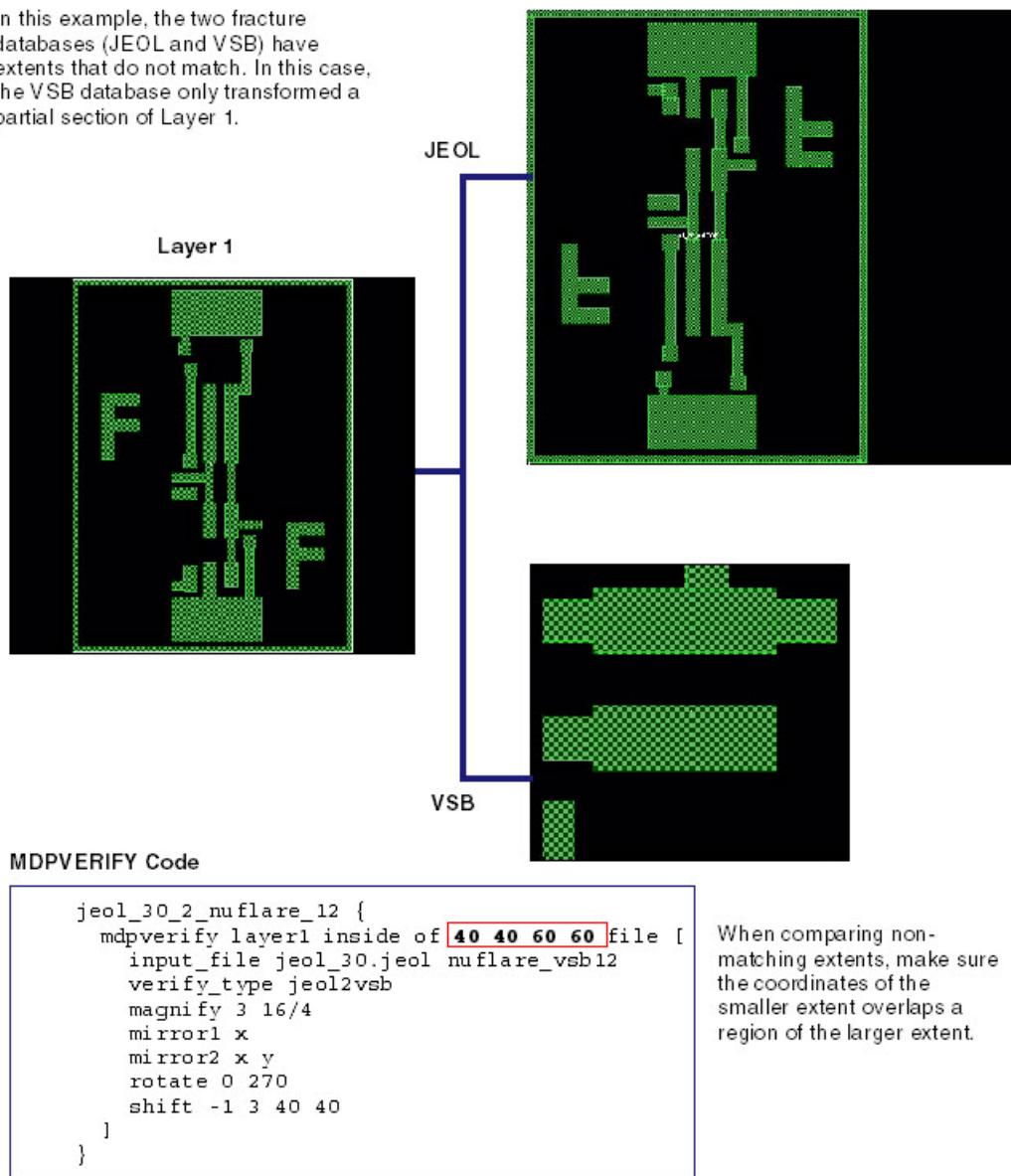
For the VSB12 (Nuflare) Fracture (note the coordinates for a smaller region in bold):

```
nuflare_12 {
    fracture nuflare Layer1 inside of 40 40 60 60 file [
        chip_directory nuflare_vsb12
        log_file_name nuflare_vsb12.log
        max_skew_approximation_error 0.005
        version 12
        conversion_address_units 0.001
        magnify 16/4
        mirror x y
        rotate 270
    ]
}
```

The results of these fracture operations are shown in [Figure 6-7](#).

Figure 6-7. Comparing Two Databases With Different Extents

In this example, the two fracture databases (JEOL and VSB) have extents that do not match. In this case, the VSB database only transformed a partial section of Layer 1.



Comparing Two VSB Job Decks

Comparisons of two Nuflare (VSB11, VSB12, VSB12i, and NUFLARE_MBF) job decks (verify_type VSBJOB2VSBJOB) have the following limitations which differentiate them from fractured-data-to-database and fractured-data-to-fractured-data verifications.

- You cannot perform any actions involving transformation or reverse tone.
- The input job decks should each have the same mask size. If they do not, or if you wish to only compare portions of the job decks, you must explicitly specify the verification region using the INSIDE OF keyword.

- All chips within the job decks must have the same conversion address units.
- It is suggested that the Precision SVRF statement in the rule file containing the MDPVERIFY statement be equal to, or an integer multiple of, the conversion address units in the job decks.

For example, if the UNIT LENGTH of your design is 1 micron (10-6 meters) and the conversion address unit of your job deck is 0.00125 microns, the precision is 800. Therefore, you should specify the SVRF Precision as 800, 1600, 3200, 6400, and so on. Any other value, including the default setting of 1000, could result in decreased performance.

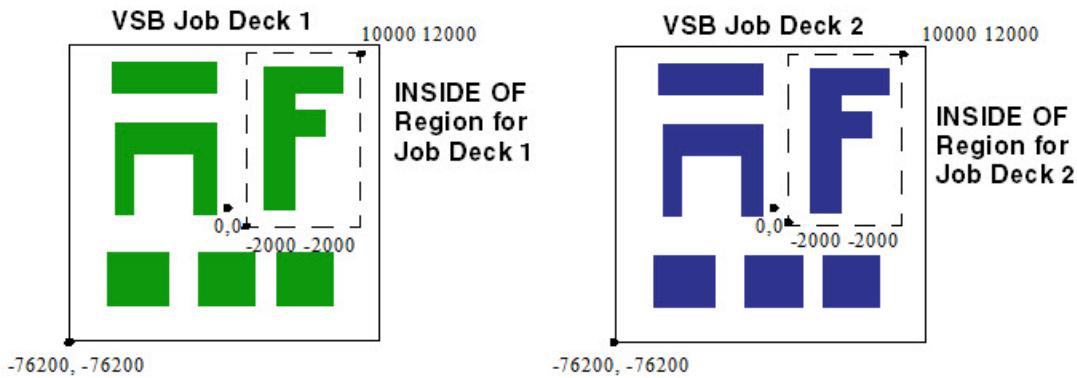
This form of verification may also require you to use the job deck specific keywords *mask1* and *mask2*. These keywords specify the names of the layouts in the job deck to be verified. You only need to specify these keywords if more than one layout exists in the job decks.

An example of the MDPVERIFY command for VSB job decks is:

```
check_vsb = MDPVERIFY layerX INSIDE_OF (-2000) (-2000) 10000 12000 FILE [
    input_file ./jobdecks/vsbJobDeck_1/ ./jobdecks/vsbJobDeck_2/
    verify_type VSBJOB2VSBJOB
    mask1 XYR203
    mask2 XYR203
    shift -76200 -76200 -76200 -76200
]
```

This example shows two VSB job decks, identical in mask size (illustrated in Figure 6-8). An INSIDE OF region has been defined in the extents of each job deck for comparison.

Figure 6-8. INSIDE OF Regions Comparing VSB Job Decks



```
check_vsb = MDPVERIFY layerX INSIDE_OF (-2000) (-2000) 10000 12000 FILE [
    input_file ./jobdecks/vsbJobDeck_1/ ./jobdecks/vsbJobDeck_2/
    verify_type VSBJOB2VSBJOB
    mask1 XYR203
    mask2 XYR203
    shift -76200 -76200 -76200 -76200
]
```

Note

 The comparison region in the previous example is specified in the job deck coordinate system, which is different from the coordinate system used by default when comparing two external data sources. Therefore, shifting is used to align job decks properly before comparison. For more details, refer to “[Avoiding Problems With Shifting](#)” on page 220.

Using Proper Shift Values

The shift keyword lets you shift the data as part of the conversion into the Calibre database format so that it aligns with the data to which it is being compared.

Some shifting is performed automatically, as the MDPVERIFY operation compensates for differences between the various fracture formats (origin adjustments).

Automatic Shift Alignment	218
Manual Shift Alignment	219

Automatic Shift Alignment

Calibre MDPverify performs automatic shift alignment when comparing external fracture data to an internal Calibre layer and you do not specify the *shift* keyword in your MDPVERIFY statement.

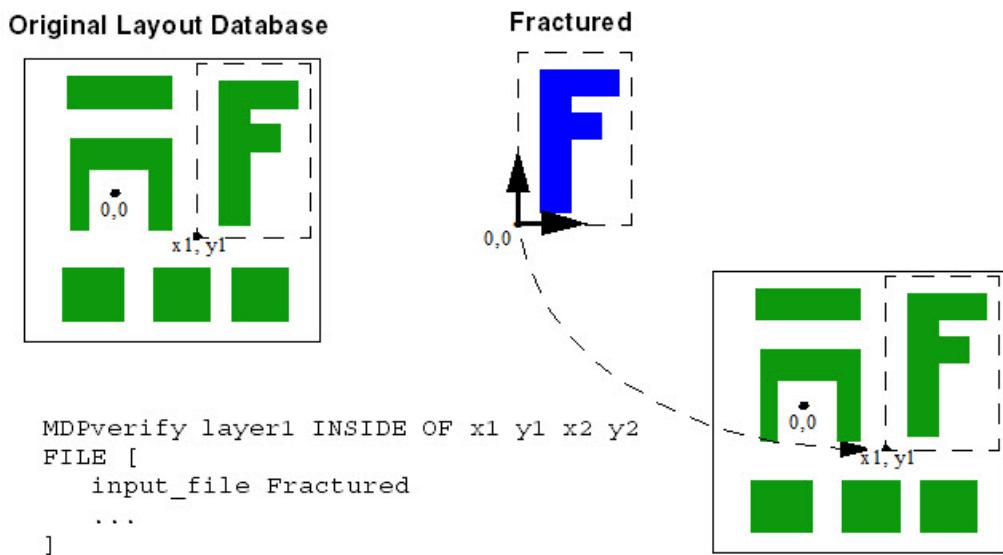
The default shift values are appropriate in the following situations:

- Comparing fractured data to layout data when the lower left corner of the verification region is the same as the lower left corner of the original fracture region.
- Comparing fractured data to fractured data when the two sets of data are the same (that is, both files are fractured versions of the exact same portion of the original layout.)

Automatic Shift with One External File

When there is one external file, the operation shifts the data by (xI, yI) , where (xI, yI) are the coordinates of the lower left corner of the verification region specified by the INSIDE OF argument.

Figure 6-9. Default Shifting with One File

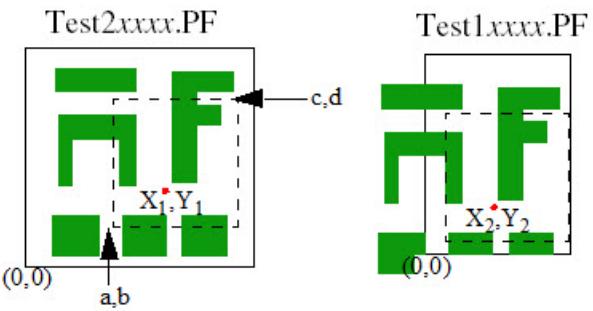


Manual Shift Alignment

The following is an example of comparing the same region in two files with different origins.

1. Identify the set of data to be shifted: For this explanation, assume one set is shifted, one is not.
2. Define the transformations necessary to force the two sets of data into the same scale and orientation. (After transformation, the coordinates for each are in terms of the Calibre database.)
3. Identify a common point, and locate its coordinates in the two different sets of transformed data as (X1,Y1) and (X2,Y2) where (X1,Y1) is in the data to be shifted.
4. Calculate the shift values as (X1-X2) and (Y1-Y2).
5. Apply the shift values to the data containing (X1,Y1).
6. Calculate the verification region using coordinates.

Figure 6-10. Calculating Shift Values



```
MDPverify layer1 INSIDE OF a b c d FILE [
    input_file Test2xxxxx.PF Test1xxxxx.PF
    verify_type MEBES2MEBES
    shift 0 0 (X1-X2) (Y1-Y2)
]
```

Avoiding Problems With Shifting

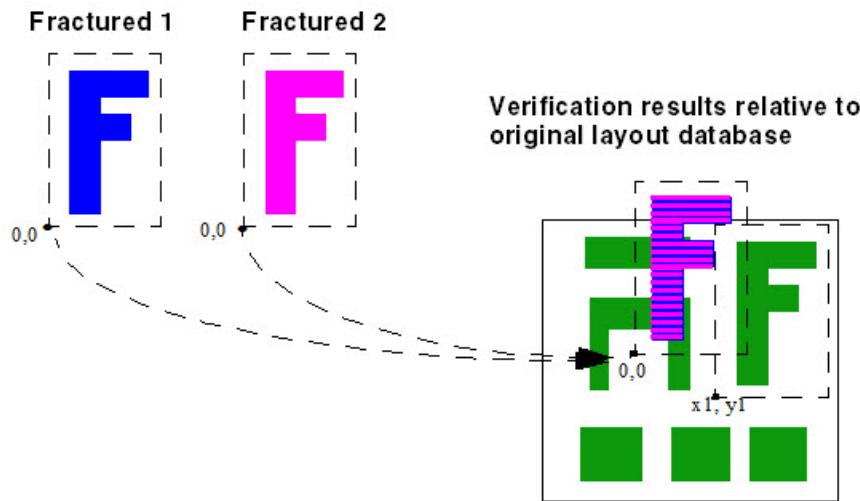
One problem you are likely to encounter related to shifting, is comparing two fractured files using the default value for shift but not the default value for the verification region (as specified using INSIDE OF).

The default behavior for shift aligns the fractured data with the database origin, while the verification region maintains its original coordinates. There are two easy solutions:

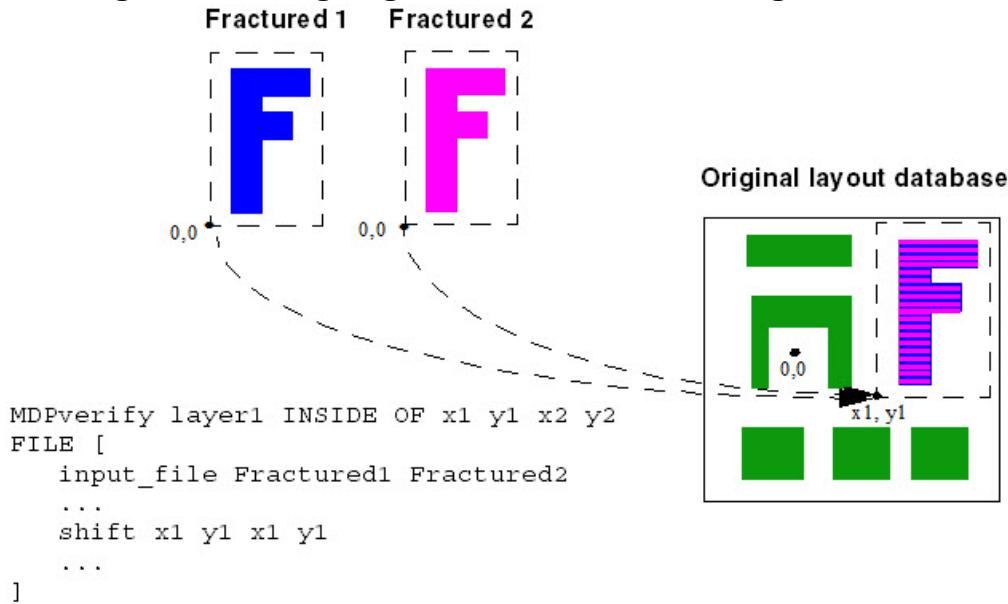
- Solution 1: Do not shift the data and do not specify a verification region. This causes the verification region to default to the extent of the fractured data. Note that you can only use this solution when the two files have the same transforms and the same extents.
- Solution 2: Specify the FRACTURE region and also shift both files by x_1, y_1 , where x_1, y_1 are the coordinates of the lower left corner of the FRACTURE region. This assumes that the verification region is specified using the same coordinate region.

Another problem you might encounter is when you want to compare two fracture files, intending to align the results with the original data for viewing or further processing using SVRF operations.

Assume you have two fractured files that represent the same portion of the original layout. Calibre MDPverify lines up the two fractured files with the database origin, verifying like against like. However, the data is not lined up properly with the origin of the original database.

Figure 6-11. Verification Region Not Aligned With Original Data

In order to overlay the errors returned from verification on the correct portion of the layout, you must shift the data by x_1, y_1 , where x_1, y_1 are the coordinates of the lower left corner of the original fracture region (which is typically the same as the verification region).

Figure 6-12. Aligning Fractured Data With Original Data

Define the Verification Region

You specify the portion of the data to be verified or compared by defining the verification region in terms of the Calibre database grid. Because all data must be converted into the Calibre database format before comparison, it is the one representation common to all data. Calibre MDPverify isolates the verification region AFTER the data is transformed and converted into

the Calibre database format. Thus, if you have set the transformation controls properly the fractured data is aligned with each other and the verification region identifies the same portion of the design in both sets of data.

To compare the entire fractured data file to the original layout database, specify the same coordinates for the verification region as those used for the fracture region when fracturing the data.

To compare a portion of the fractured data file to the original layout database, specify the coordinates relative to the original layout. When you do so, you must specify a shift value for the fractured data. For more information on shifting, refer to “[Using Proper Shift Values](#).”

To compare one set of fractured data to another set of fractured data, specify the coordinates relative to the Calibre database coordinates.

Examples

The following contains sample rule files used to perform several comparisons. Each example begins with the same layout database. However, the transformations applied when fracturing the data may vary.

Example 1 — MEBES to an Internal Calibre Layer	223
Example 2 — MEBES to MEBES	224
Example 3 — MEBES to JEOL	225

Example 1 — MEBES to an Internal Calibre Layer

This example creates a MEBES pattern file from a layer, then compares it to the original layer.

```

LAYOUT SYSTEM GDSII
LAYOUT PATH "sample.gds"
LAYOUT PRIMARY "Layout1"

PRECISION 1000
DRC MAXIMUM RESULTS ALL

// Set DRC MAXIMUM RESULTS to ALL to ensure that all
// differences are saved to the results database

DRC RESULTS DATABASE "Example1.gds" GDSII PSEUDO

LAYER orig6 6

// Include DRC SUMMARY REPORT to ensure that
// you get an accurate count of the differences

fracture6 {FRACTURE MEBES orig6 INSIDE OF (-880) (-670) (-400) (-270)
FILE [
mode 5
magnify 4
compaction_stripes 64
address_size 0.020
file_name TEST1XXXX.PF
]
}
// Use the same transformation parameters settings for
// Fracture and MDPverify. MDPverify then reverses the transformation.

check_gds = MDPverify orig6 INSIDE OF (-880) (-670) (-400) (-270) FILE [
input_file TEST1XXXX.PF
verify_type MEBES2DB
magnify 4
]
// Output the results plus the original data to the GDS database.
// The results of MDPverify are all geometry that is in
// either the layout database or the fractured data but not in both.

orig6 {COPY orig6 } DRC CHECK MAP orig6 6
check_gds {COPY check_gds} DRC CHECK MAP check_gds 29

```

Example 2 — MEBES to MEBES

This example creates a MEBES pattern file from a layer, then compares it to the MEBES file generated in Example 1 from the same layer.

```
LAYOUT SYSTEM GDSII
LAYOUT PATH "sample.gds"
LAYOUT PRIMARY "Layout1"

PRECISION 1000
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "Example2.gds" GDSII PSEUDO

LAYER orig6 6

// When fracturing, Rotate and mirror for mask making purposes
// scale by 5x

fracture2 {FRACTURE MEBES orig6 INSIDE OF (-880) (-670) (-400) (-270)
    FILE [
        mode 5
        rotate 90
        mirror x
        magnify 5
        compaction_stripes 64
        address_size 0.010
        file_name TEST2XXXX.PF
    ]
}

//Verify against the MEBES file created in Example 1 from same Design

check_mebes = MDPverify orig6 INSIDE OF (-880) (-670) (-400) (-270) FILE [
    input_file TEST2XXXX.PF TEST1XXXX.PF
    verify_type MEBES2MEBES
    magnify 5 4
    rotate 90 0
    mirror1 x
]

orig6 {COPY orig6 } DRC CHECK MAP orig6 6
check_mebes {COPY check_mebes} DRC CHECK MAP check_mebes 30
```

Example 3 — MEBES to JEOL

This example creates a JEOL pattern file from a GDS layer, then compares it to the MEBES file generated in Example 1 from the same GDS layer.

```
LAYOUT SYSTEM GDSII
LAYOUT PATH "sample.gds"
LAYOUT PRIMARY "Layout1"

PRECISION 1000
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "Example3.gds" GDSII PSEUDO

LAYER orig6 6

// When fracturing, Rotate and mirror for mask making purposes
// scale by 4x

fracture3 {FRACTURE JEOL orig6 INSIDE OF (-880) (-670) (-400) (-270)
    FILE [
        rotate 180
        mirror y
        magnify 4
        fracture_units 100
        file_name TEST3.JF
    ]
}

//Verify against the MEBES file created in Example 1 from same Design

check_jeol_mebes = MDPverify orig6 INSIDE OF (-880) (-670) (-400) (-270)
    FILE [
        input_file TEST1XXXX.PF TEST3.JF
        verify_type MEBES2JEOL
        magnify 4 4
        rotate 0 180
        mirror2 y
    ]

orig6 {COPY orig6 } DRC CHECK MAP orig6 6
check_jeol_mebes {COPY check_jeol_mebes} DRC CHECK MAP check_jeol_mebes 32
```

Chapter 7

Calibre MDPmerge

A Calibre MDPmerge operation reads a job deck and merges individual chips and chip placements into a single chip and placement based on certain criteria. The output is a new job deck consisting of the merged chips and copies of chips that did not meet the merge criteria.

You can use Calibre MDPmerge to do the following:

- Optimize the throughput of the mask writing machine.
- Optimize the parameter control like registration and CD.
- Enable the proximity correction for adjacent placements of individual chips based on certain criteria (for example, the distance from each other, write sequence).

Currently, Calibre MDPmerge supports only job decks for VSB11 format with some restrictions.

All chip placements of a particular drawing group are merged if any two individual placements in the drawing group are closer than the user-specified distance.

Multithreaded Operations	227
Calibre MDPmerge Limitations.....	228
Supported Job Deck Values	228
MDPMERGE	229
View the Calibre MDPmerge Results	232

Multithreaded Operations

For multithreaded operations, Calibre MDPmerge uses one license for the first CPU, and an additional one license per four CPUs. There is no substitute license for Calibre MDPmerge.

Note

 When performing multithreaded processing, Calibre MDPmerge memory consumption is dependent on the number of CPUs you specify. The amount of memory Calibre MDPmerge uses increases at approximately the same rate you add CPUs (for example, specifying 16 CPUs can use double the amount of memory of eight CPUs).

Calibre MDPmerge Limitations

A number of limitations apply to Calibre MDPmerge.

- Calibre MDPmerge supports zero and one mirror options in the drawing script (job deck).
- Calibre MDPmerge provides no support for object macros. Parameter macros are passed through into the output job deck.
- Chips in a drawing group must have the same address units and cell subfield size in order for them to be merged.

Supported Job Deck Values

Calibre MDPmerge recommends that you specify the *scale* value for **put**, **multiput**, or **matrixput** commands in your VSB job deck within a particular range.

```
0.5 <= scale <= 1.0      \\recommended
```

and only supports *scale* values in the range:

```
0.2 <= scale <= 1.0      \\supported
```

Calibre MDPmerge issues a *warning* in the transcript and continues the run if you specify a *scale* value outside the recommended range, for example:

```
WARNING: chip scale value is out of the recommended range of [0.5,1.0]:  
scale = <value> for put/multiput/matrixput command at line <n> in  
file <filename>.
```

Calibre MDPmerge issues this warning for each *scale* value outside of the recommended range.

Calibre MDPmerge issues an error and skips the MDPmerge operation if you specify a scale value outside the supported range, for example:

```
ERROR: chip scale value is out of the supported range of [0.2,1.0]:  
scale = <value> for put/multiput/matrixput command at line <n> in  
file <filename>.  
Failed to read and parse VSB job deck
```

Calibre MDPmerge issues this error only once, even if there are multiple violations.

MDPMERGE

SVRF mask data preparation command that reads a job deck and merges individual chips and chip placements into a single chip and chip placement.

Usage

```
MDPMERGE format FILE {parameter_block_name | '['  
    input_dir dir1  
    output_dir dir2  
    layout_name name_of_input_layout  
    merge_dis distance  
    [frame_max_data max_frame_size]  
    [common_max_data max_file_size]  
    [frame_bde_max_data max_frame_bde_size]  
    [common_bde_max_data max_common_bde_size]  
    [common_cell_max value]  
    [cell_def_max value]  
    [cell_loc_max value]  
    [frame_width width]  
    ']'  
}  
}
```

Description

Calibre MDPmerge reads data externally from the disc and writes output to the disc, bypassing the Calibre database. The Calibre MDPmerge operation is a Standard Verification Rule Format (SVRF) operation. Consequently, as part of the Calibre rule file definition for calling the Calibre MDPmerge command, you must specify a layout database file (normally, a dummy file) containing at least one geometry. Additionally, you must specify an output layout database file, which is empty when no layers are written out.

Arguments

- ***format***
A required argument specifying the format of the file. Only VSB11 is supported.
- **FILE *parameter_block_name***
A required keyword that specifies the output definition parameters are contained in a reusable parameter block defined using the **LITHO FILE** SVRF statement.
- **FILE '[' ... ']'**
A required keyword that specifies that the output definition parameters are contained inline within the square brackets ([]). Square brackets must surround all operation arguments

appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:

- Keywords and parameters must be on lines STRICTLY BETWEEN the left and right brackets (that is, they cannot be on the same line).
 - Argument text cannot contain either a left ([) or right bracket (]).
 - You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.
- **input_dir *dir1***
A required keyword that specifies the directory path where the layout.ini file of the input job deck resides.
 - **output_dir *dir2***
A required keyword that specifies the directory path where the output job deck is stored. If no merge is needed for the input, then the output job deck is not created.
 - **layout_name *name_of_input_layout***
A required keyword that specifies the name of the input layout. The value specified for *name_of_input_layout* must match the layout_name parameter in the *layout.ini* file in the *input_dir* directory. While many mask layouts can be specified in the VSB job deck, Calibre MDPmerge processes only one layout at a time.
 - **merge_dis *distance***
A required keyword that specifies a distance (in microns at mask scale) between any two chips in a given drawing group. This can be a floating point value in microns, such as 40.04. If the distance between two chips is less than the *distance* value, then all chips of the corresponding drawing group are merged into one chip.
The merge_dis behavior is enforced in both the x- and y-directions.
 - **frame_max_data *max_frame_size***
An optional keyword that specifies the maximum size of *frame.X* files in units of megabytes (MBs). The *max_frame_size* value must be less than 8191 MBs. The default is 512 MBs.
 - **common_max_datamax_file_size**
An optional keyword that specifies the maximum size of the *common.chp* file in units of MBs. The *max_file_size* value must be less than 8191 MBs. The default is 512 MBs.
 - **frame_bde_max_data *max_frame_bde_size***
An optional keyword that specifies the maximum Boundary Data Extent (BDE) size of *frame.X* files in units of MBs for the resultant chip from MDPMERGE. See VSB11 standard for more information on BDE. The value of this parameter should not exceed 8191MB. The default value is 512 MBs.

- **common_bde_max_data *max_common_bde_size***
An optional keyword that specifies the maximum BDE size of common.chp file in units of MBs for the resultant chip from MDPMERGE. See the VSB11 standard for more information on BDE. The value of this parameter should not exceed 8191MB. The default value is 512 MBs.
- **common_cell_maxvalue**
An optional keyword that specifies the maximum number of common cell definitions allowed in the common frame. The default is 5000000.
- **cell_def_maxvalue**
An optional keyword that specifies the maximum number of local cell definitions allowed in the *frame.X* file. The default is 5000000.
- **cell_loc_maxvalue**
An optional keyword that specifies the maximum number of cell locations allowed in the *frame.X* file. The default is 5000000.
- **frame_width *width***
An optional keyword used to define the maximum width of each frame. The width is specified in microns at mask scale. The default is 1024 microns.
Note that this value is actually an upper limit on the frame width. The actual frame width varies, based on the values of frame_max_data and frame_bde_max_data.

Examples

Figure 7-1 illustrates an example Calibre MDPmerge run.

Figure 7-1. Calibre MDPmerge Example**Job Deck Structure Before Running Calibre MDPmerge**

```
chip.ini
layer_1.VSB
layer_12.VSB
layer_6.VSB
layer_7.VSB
layout
layout.ini
```

SVRF Rule File

You must supply an input layout file

```
LAYOUTPATH "./input.gds"
LAYOUT SYSTEM GDSII
LAYOUT PRIMARY "*"
DRC RESULTS DATABASE "merge_results.gds" GDSII PSEUDO
DRC SUMMARY REPORT "merge_results.txt"
PRECISION 10000
DRC MAXIMUM RESULTS ALL
LAYER orig1 1
LAYER orig6 6
LAYER orig7 7
LAYER orig12 12
mergel {MDPMERGE vsb11 FILE [
    input_dir "/scratch1/data"
    output_dir "/output.VSB.JB"
    merge_dis 40
    layout_name "MERGE_TEST"
]}
```

Calibre MDPmerge Output (contents of *output.VSB.JB*)

```
drwxr-xr-x  2 user      512 Jun 10 13:18 MDPmerge_chip_1
-rw-rw-r--  1 user     120 Jun 10 13:18 chip.ini
drwxr-xr-x  2 user      512 Jun 10 13:06 layout
-rw-rw-r--  1 user      93 Jun 10 13:18 layout.ini
```

The output from Calibre MDPmerge is written such that the *chip_name* parameter value is in uppercase in the *chip.ini* and *chip.cnf* files, and the *chip_dir*, *chip_id*, and *locator_id* values are in lowercase.

Calibre MDPmerge does not reformat or modify any files related to a chip in the input job deck that does not need to be merged. These unaffected files are copied directly to the output job deck without modification.

View the Calibre MDPmerge Results

You can use the Calibre MDPview tool for job viewing and visual verification. Calibre MDPview supports a number of job deck parameters and commands for VSB11.

Parameters

- chip_data_format
- chip_dir
- chip_name
- class
- layout_name
- layout_system
- mask_mirror
- mask_size
- mask_size_x
- mask_size_y
- script
- unit_length

Locate

- class
- matrixput
- put
- mirror 0|1
- multiput

Macros

- parameter macros

Drawing

- draw
- xdraw with parameter macros
- refer with parameter macros

Miscellaneous

- double slash (//) for comments

Chapter 8

Calibre MDPstat

The MDPSTAT SVRF operation analyzes Hitachi, JEOL, Micronic and VSB formatted data based on the following criteria, allowing you to assess the quality of the fracture output.

- Small outside trapezoid detection — detects critically-dimensioned trapezoids whose edges have a specified amount of exposed length.
- Split CD detection — detects critically-dimensioned polygons that have been split lengthwise into trapezoids.

Calibre MDPstat writes statistical data to the Calibre transcript and generates a new layer containing the applicable geometries.

Calibre MDPstat Reporting	235
MDPSTAT	237

Calibre MDPstat Reporting

Calibre MDPstat reports its statistical information in the Calibre nmDRC-H Executive Module section of the transcript.

```
s0 = MDPSTAT SMALLOUTSIDETRAP 0.15 0.5 INSIDE OF LAYER POLY (LITHO)FILE sf
s1 = MDPSTAT SPLITCD 0.5 1 INSIDE OF LAYER POLY (LITHO)FILE sf
...
*** Small figure distribution.***
      Upper limit      Count.
      0.150000          6
*** Split CD distribution.***
      W limit          L limit      Count.
      0.500000          1.000000          4
s0 (FLAT TYP=1 CFG=1 ECT=11 OCT=7 SRT=1 CMP=F MPN=5 VFC=F)
s1 (FLAT TYP=1 CFG=1 ECT=6 OCT=6 SRT=1 CMP=F MPN=2 VFC=F)

...
s0:::<1> = COPY s0
-----
...
DRC RuleCheck s0 COMPLETED. Number of Results = 5 (5)
...
s1:::<1> = COPY s1
-----
...
DRC RuleCheck s1 COMPLETED. Number of Results = 2 (2)
...
```

The difference between the Count value in the upper section and the Number of Results values in the DRC RuleCheck section is due to the merging of the data on the layers containing the MDPSTAT results.

MDPSTAT

SVRF mask data preparation command used to generate statistical information.

Usage

```
MDPSTAT {SMALLTRAP s [l] | SMALLOUTSIDETRAP s [e] [l] | SPLITCD w l}
{INSIDE OF x1 y1 x2 y2 | INSIDE OF LAYER layer1 | INSIDE OF EXTENT}
[FILENAME name_of_file] FILE {parameter_block_name} | '[
    input_file file
    verify_type {JEOL2DB | VSB2DB | HITACHI2DB | MICRONIC2DB |
        VBOASIS2DB}
    [maximum_output_count limit]
    [magnify value]
    [rotate {0 | 90 | 180 | 270}]
    [mirror1 {x | y | x y}]
    [shift x y [x y]]
    ']'
}
```

Description

The MDPSTAT SVRF operation analyzes formatted data based on specified criteria (small outside trapezoid detection and split CD detection). It then writes statistical data to the Calibre transcript and generates a new layer containing the applicable geometries.

Arguments

- {**SMALLTRAP** *s* [*l*] | **SMALLOUTSIDETRAP** *s* [*e*] [*l*] | **SPLITCD** *w* *l*}

A required keyword and value set specifying the statistical information to generate. Choose one of the following:

SMALLTRAP *s* *l* — Reports all trapezoids that are smaller than the specified value *s*. A trapezoid is considered small if the smaller dimension of its rectangular extent is less than *s*. If *l* is specified, then SMALLTRAP reports all small trapezoids with lengths greater than this value. The default value of *l* is 0, reporting all small trapezoids.

SMALLOUTSIDETRAP *s* *e* *l* — Detects and outputs trapezoids where the smallest dimension of its rectangular extent is less than *s*. To determine whether a trapezoid is a SMALLOUTSIDETRAP, the following rules apply:

- If any two consecutive sides are fully exposed, the trapezoid is always considered outside.
- If two consecutive sides are not fully exposed, then Calibre MDPstat checks if the trapezoid's cumulative amount of exposed length along one of the two longest sides of the rectangular extent is greater than *e*. This rule also checks if the longer side of this sliver exceeds the length criteria, *l*. This algorithm is exact for rectangles.

Note

 This measurement scheme may cause missed detections in rare instances, such as when the sides of an extent corresponding to the outside edges of a trapezoid coincidentally abuts another feature.

s — Specified in user units in the Calibre coordinate system.

e — Specified in user units in the Calibre coordinate system, with a default value of twice the value of *s* ($e=2*s$).

If *l* is specified, then SMALLOUTSIDETRAP reports all small-outside trapezoids with lengths greater than the specified value. The default value of *l* is 0, which reports all small-outside trapezoids. Both *l* and *e* must be specified together.

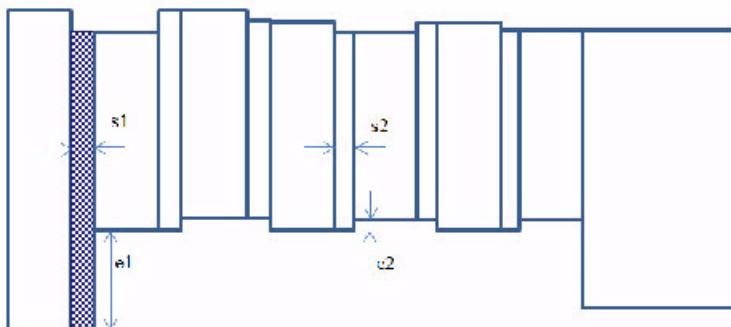
SPLITCD *w l* — Detects and outputs trapezoids where The shorter side of the rectangular segments of a polygon is less than *w* and the longer side of each rectangular segment is greater than *l*, and an edge of a trapezoid is inside of the polygon and parallel to the longer side of the rectangular polygon.

w — specified in user units in the Calibre coordinate system

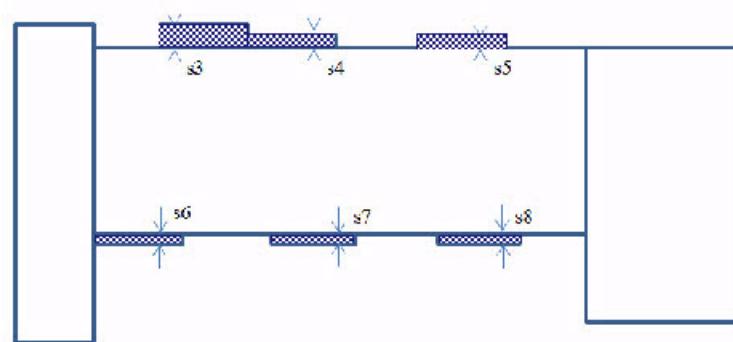
l — specified in user units in the Calibre coordinate system.

The following figures illustrate examples of SMALLOUTSIDETRAP and SPLITCD.

Figure 8-1. SMALLOUTSIDETRAP Examples



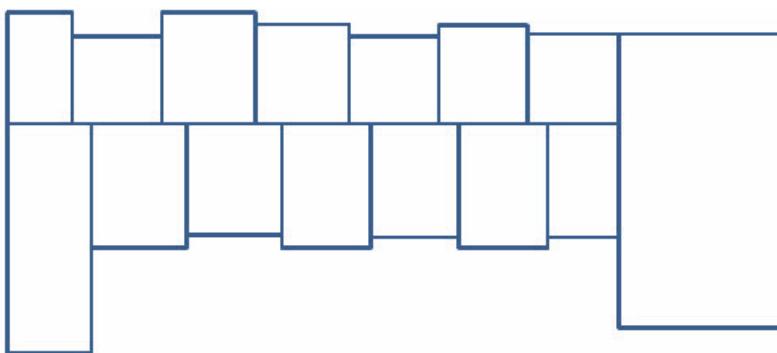
The shaded area is selected by MDPSTAT SMALLOUTSIDETRAP. In this case, $s_1 < s$ and $e_1 > e$. Small inside trapezoids with an exposed edge $< e$ (such as e_2) are not selected.



The shaded areas are selected by MDPSTAT SMALLOUTSIDETRAP. All 6 small outside traps satisfy the requirement that one edge length is $< s$. Additionally, these trapezoids also satisfy the criteria that they are “completely outside” or “outside corner” trapezoids.

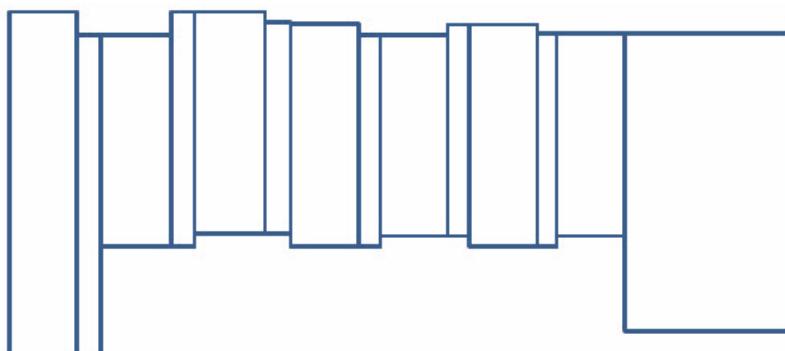
Shapes s_5 , s_7 , and s_8 are “completely outside” trapezoids, where three edges are completely exposed.

Shapes s_3 , s_4 , and s_6 are “outside corner” trapezoids, where two adjacent edges are completely exposed and the other two adjacent edges are completely covered.

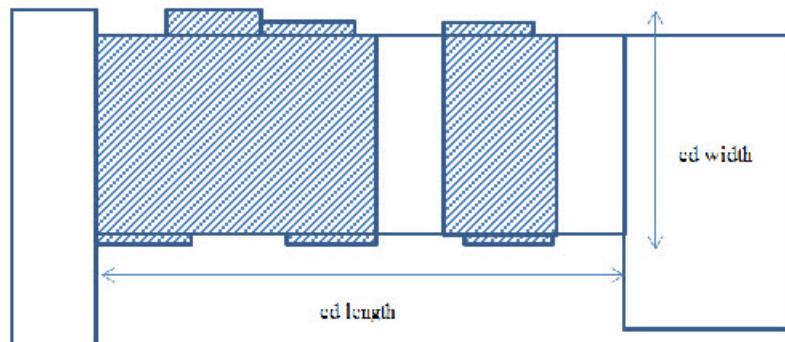


No areas are selected in this configuration by MDP SMALLOUTSIDETRAP. However, this configuration is flagged by MDPSTAT SPLITCD.

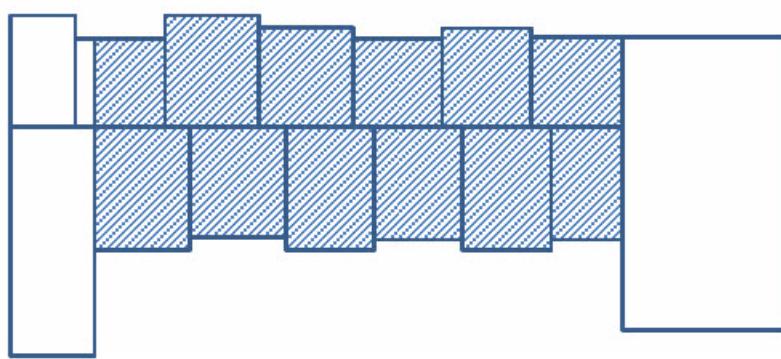
Figure 8-2. SPLITCD Examples



No areas are selected by MDPSTAT SPLITCD.



The shaded areas are selected by MDPSTAT SPLITCD. This CD is identified as a critical CD because the envelope of its trapezoids satisfies the width and length criteria set by the user in the MDPSTAT SPLITCD command. The shaded areas are selected because they are the portions of the CD polygon where the CD is split along its length.



No areas are selected by MDPSTAT SPLITCD.

- {**INSIDE OF***x1 y1 x2 y2 | INSIDE OF LAYER* *layerI | INSIDE OF EXTENT*}

A required keyword and value set used to define the verification region for the MDPstat operation. This argument must be one of:

- **INSIDE OF** *x1 y1 x2 y2*

Defines the verification region as a rectangle with the specified coordinates. You must write coordinates relative to the origin of the Calibre database, and enclose negative coordinates in parentheses (). For example:

```
INSIDE OF (-123) 123 (-12) 456
```

- **INSIDE OF LAYER *layer1***
Defines the verification region by supplying a layer whose extent defines the region.
- **INSIDE OF EXTENT**
When INSIDE OF EXTENT is specified, the extent of geometries in the input file in the FILE *parameters* section is used. Note that when INSIDE OF EXTENT is specified, no transforms can be specified.
- **FILENAME *name_of_file***
An optional keyword set used to specify the name of the fracture output file.
name_of_file — A user-supplied filename that consisting of alpha-numeric characters, underscores (_), and at most one period (.), which is part of the extension. You can also specify this argument as a variable.
This keyword is mutually exclusive from the **file_name** parameter; Calibre generates an error if you specify both for any one FRACTURE operation.
- **FILE *parameter_block_name***
A required keyword used to specify that the output definition parameters are contained in a reusable parameter block defined using the **LITHO FILE** SVRF statement.
- **FILE[' *inline_parameters* ']**
A required keyword that specifies that the output definition parameters are contained inline within the square brackets ([]). Square brackets must surround all operation arguments appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:
 - Keywords and parameters must be on lines strictly between the left and right brackets (that is, they cannot be on the same line).
 - Argument text cannot contain either a left ([) or right bracket (]).
 - You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.
- **input_file *file***
A required keyword and value set that specifies the pathname for the JEOL, Micronic, Hitachi, OASIS, and VSB formatted data to be verified. The value specified for file must be a pathname to the file.
- **verify_type{JEOL2DB | VSB2DB | HITACHI2DB | MICRONIC2DB | VBOASIS2DB}**
A required keyword and value pair indicating the input file format of the file to be analyzed. Choose one of the following:
 - JEOL2DB — JEOL
 - VSB2DB — VSB11, VSB12, VSB12i, or NUFLARE_MBF

HITACHI2DB — Hitachi

MICRONIC2DB — Micronic

VBOASIS2DB — OASIS

The format of *input_file* must match the specified *verify_type*.

- maximum_output_count {2000000 | all | limit}

An optional keyword and value pair defining the maximum number of output polygons Calibre MDPstat generates on the output layer.

2000000 — Default setting.

all — Instructs Calibre MDPstat to output all polygons.

limit — An integer specifying the maximum number of polygons to output.

This is useful for early detection of errors that cause a large number of output polygons to be generated, such as bad fracture parameter selection.

This keyword does not limit the generation of statistics written to the transcript. In other words; the run will complete checking the entire area specified, but the number of polygons it outputs is limited by the maximum_output_count value.

- magnify {1 | value | *numerator/denominator*}

An optional keyword and value set instructing Calibre MDPstat to apply an *inverse* magnification to the file before performing the analysis. The value can be any double floating point numbers.

This option magnifies the external file by 1/value.

1 — Default setting; no magnification.

value — The amount of magnification; a double floating point number.

numerator/denominator — The amount of magnification in fractional form, where *numerator* and *denominator* are integers, and the slash (/) must be specified.

- rotate {0 | 90 | 180 | 270}

An optional keyword instructing MDPSTAT to apply an *inverse* rotation to the file before performing the analysis.

Rotation is in the clockwise direction (the inverse of the FRACTURE rotation, which is counter-clockwise). When using rotate to undo a rotation that was performed when generating the fractured file, the rotate value must match the one used in the FRACTURE command.

- mirror1 {x | y | x y}

An optional keyword instructing MDPSTAT to apply an *inverse* mirror to the file before performing the analysis. Mirroring is across the axis running through the center of the region. The mirror values must match the mirror values used in the FRACTURE command.

When you specify both **mirror1** and **rotate** in the same MDPSTAT operation, mirror is always applied after rotation, regardless of their order in the rule file.

The default is to not perform any mirroring.

Note

 The intent behind Calibre MDPstat is to reverse the sequence of transformations applied when fracturing. Because of this, when mirror, rotate, or magnify keywords are used in the same MDPSTAT operation, the order of application (magnify - rotate - mirror) is used, regardless of their order in the rule file. This is the reverse of the order used when fracturing.

- shift $x\ y[x\ y]$

An optional keyword instructing MDPSTAT to shift the output data to the given coordinates.

If you do not specify a shift value, then the operation shifts the data by (x_1, y_1) , where (x_1, y_1) are the coordinates of the lower-left corner of the measurement region in the Calibre Hierarchical Database coordinate system. For example, if you want the resulting output of the command to occupy (x_1, y_1) to (x_2, y_2) , then you should set the shift values to (x_1, y_1) .

Chapter 9

Calibre Rule-Based MPC and MDP Utility Commands

Calibre MDP provides a number of utility commands relating to mask process correction (MPC), OASIS bin injection, and computing the extent of an OASIS layout, and generating input for data-to-database detection software.

- Calibre MPCpro provides you with the ability to apply pattern-based mask process corrections to your design data prior to fracturing. This includes the correction of linearity and proximity effects as well as longer range processing effects such as fogging and develop- and etch-loading. The functionality provided includes table-driven biasing (OPCBIAS SVRF statement) and the DENSITY CONVOLVE and MDP MAPSIZE statements.
- The MDP CHECKMAP SVRF statement is used primarily for OASIS bin injection.
- The MDP OASIS_EXTENT SVRF statement computes the extent of an OASIS layout.
- The MDP DATAPREP SVRF statement is used to generates input for data-to-database detection software.

Calibre MPCpro Workflow	245
DENSITY CONVOLVE SVRF Statement	248
DENSITY CONVOLVE	250
MDP MAPSIZE SVRF Statement	267
MDP MAPSIZE	270
MDP CHECKMAP SVRF Statement	272
MDP CHECKMAP	273
MDP OASIS_EXTENT SVRF Statement	276
MDP OASIS_EXTENT	277
MDP DATAPREP SVRF Statement	281
MDP DATAPREP	282

Calibre MPCpro Workflow

Mask CD (Critical Dimension) errors are caused by various types of effects including e-beam blur, electron scattering, and etch loading. These effects are caused by different physical mechanisms, and they also affect mask CD errors on very different length scales.

For example, e-beam blur and electron forward scattering affect mask CDs at very small dimensions that are typically far below a micrometer. For other effects, like fogging and etch, loading typically has a range of influence in the millimeter range.

The influences of these effects on mask CDs highly depend on pattern size and pattern density. These dependencies are leveraged in Calibre MPCpro to correct for systematic mask CD errors.

Calibre MPCpro corrects mask CD errors using the following flow:

1. Perform long-range correction:
 - a. Use the jobToOasis utility to convert the job deck to an OASIS file (job deck smashing). See the *Calibre MDPview User's Manual* for complete information on the jobToOasis utility.
 - b. In the SVRF, use DENSITY CONVOLVE to apply convolutions. You can use direct data entry for very large dense masks, or HDB-entry if the mask is small or sparse.
 - c. Use MDP MAPSIZE to size the data within each of the density windows (for example, sparse areas are sized up more, dense areas are sized down).
2. Perform short-range correction:
 - a. Use table-driven OPC biasing (OPCBIAS SVRF command) to size the edge fragment up or down depending on how close the adjacent shape is. OPCBIAS is used in ESVRF in MDP EMBED or fracture, and is highly-scalable. It can handle large designs with low RAM-enabled direct-data-entry and ESVRF. Refer to the *Calibre Rule-Based OPC User's and Reference Manual* for a complete description.

The following is an example demonstrates an entire flow for Calibre MPCpro.

```
layout system oasis
layout path "./oasis/dummy.oas"
layout primary "*"
precision 1000
drc maximum results all
drc results database "./oasis_out/frac_mic_out_with_corrections.oas"
oasis pseudo
layer dummy 0

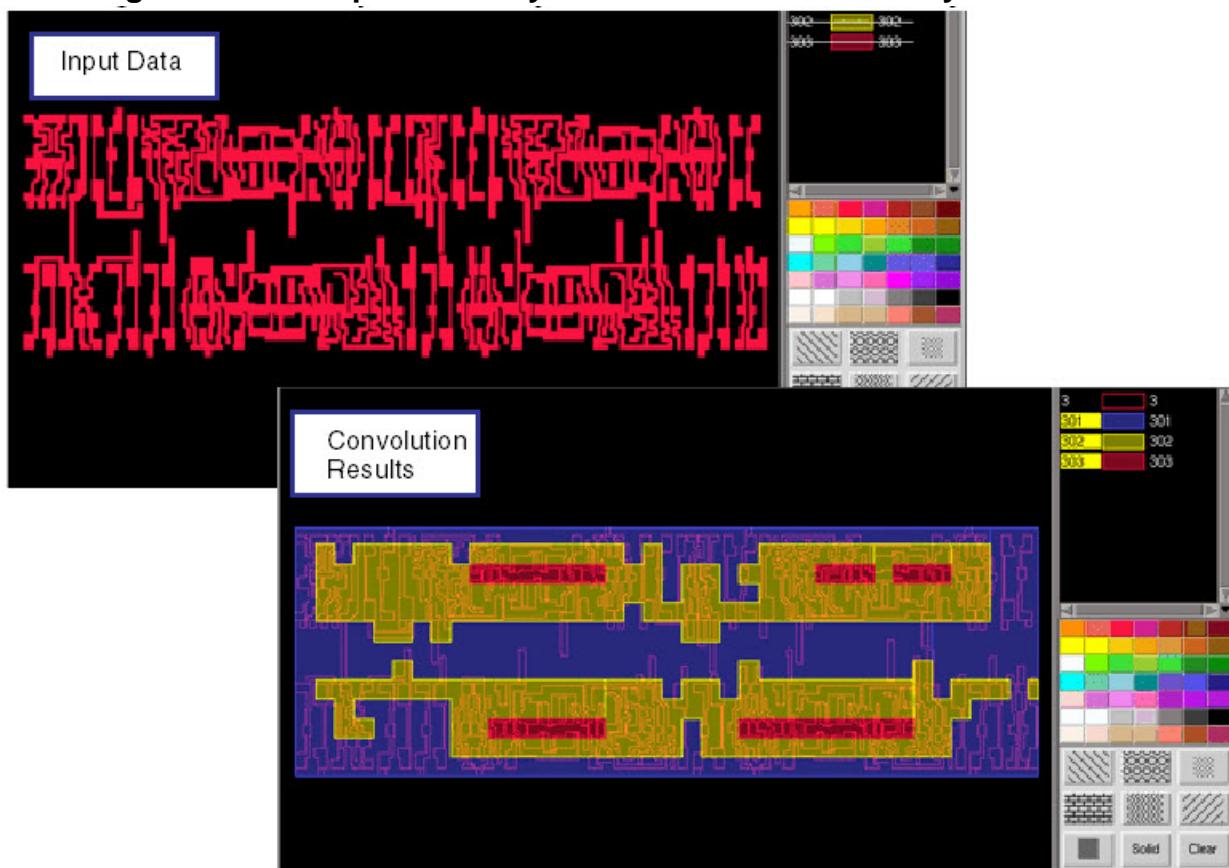
mic_frac { fracture micronic dummy inside of 0 0 2054 2182 file [
    version 1.6
    file_name "./frac/micronic_with_corrections_and_hdb_bypass.la"
    log_file_name "./log \
        frac_log_frac_mic_with_corrections_and_hdb_bypass.log"
    cd internal .8 1.6
    computation_mode section
    vboasis_path "./oasis_out/micronic.la.oas"
    svrf_layer_name prime 0
<SVRFSTART>
    mic_frac_corrected {
        opcbias prime minbiaslength 0.150
        space >= 0.150 < 0.160 opposite extended 0.150 width < 0.700 \
            move 0.012
        space >= 0.160 < 1.200 opposite extended 0.150 width < 0.700 \
            move 0.010
        space >= 1.200           opposite extended 0.150 width < 0.700 \
            move 0.007
        space >= 0.150 < 0.160 opposite extended 0.150 width >= 0.700 \
            move 0.013
        space >= 0.160 < 1.200 opposite extended 0.150 width >= 0.700 \
            move 0.011
        space >= 1.200           opposite extended 0.150 width >= 0.700 \
            move 0.008
    }
<SVRFEND>
]
```

DENSITY CONVOLVE SVRF Statement

The DENSITY CONVOLVE SVRF statement allows you to create a representation of your input layer that shows density values after applying convolution algorithms. Convolution is a mathematical procedure that produces a weighted moving average of two functions, in this case it evaluates the density value of a density window by taking into consideration the values of those windows surrounding it. Therefore the density values are “spread out” to nearby density windows.

Figure 9-1 shows two images, one is the input layer and the second displays the layers showing the density convolution.

Figure 9-1. Examples of a Layer Before and After Density Convolution



DENSITY CONVOLVE behaves similarly to the [DENSITY SVRF statement](#), in that it creates a density grid across the extent of your data. One major difference is that in the creation of the grid, no overlapping sections are allowed, therefore the step size is equivalent to the window size. Calibre calculates a density value for each window based on a specified *density_expression*.

The tool then applies a convolution algorithm to the density value for each window. You specify the settings for the algorithm, which consists of one or more two-dimensional Gaussian kernels, within the FILE argument.

The output of the DENSITY CONVOLVE SVRF statement is a layer containing a merged representation of all the windows where the density value after convolution, satisfies the specified *constraint*.

Typically you write multiple DENSITY CONVOLVE statements to show a range of density values, making use of the [LITHO FILE](#) statement, for example:

```
LITHO FILE convolve_parameters [
    SCALE 0.2 GAUSS 1
    SCALE 0.8 GAUSS 48
]

convolve out 1 = DENSITY CONVOLVE 3 < 0.3 TRUNCATE WINDOW 25
FILE convolve_parameters

convolve out 2 = DENSITY CONVOLVE 3 >= 0.3 < 0.4 TRUNCATE WINDOW 25
FILE convolve_parameters

convolve out 3 = DENSITY CONVOLVE 3 >= 0.4 TRUNCATE WINDOW 25
FILE convolve_parameters
```

Calibre executes DENSITY CONVOLVE statements concurrently when they use the same settings for the *input_layer*, WINDOW, INSIDE OF, and TRUNCATE/WRAP keywords. This allows Calibre to compute the initial density values only once for concurrent DENSITY CONVOLVE statements, saving on processing time.

Density Convolve can directly read certain input formats such as OASIS and write directly to a text file without using an RDB; however, you need to supply dummy values to the required input and output statements.

Note

 This method is only recommended for post-tapeout operations, such as mask job decks.

For Layout Path, provide an empty OASIS file. One way to create one is in a Calibre viewer, select **File > NewLayout**, set the precision to match the Precision statement of the rule file, and save under a name such as “*empty_10000.oas*”.

For the output, either set DRC Results Database to a file that is not otherwise used, or add a Layout Place Cell statement.

Refer to the description of the [DENSITY CONVOLVE](#) SVRF command for detailed information about the syntax.

DENSITY CONVOLVE [250](#)

DENSITY CONVOLVE

SVRF mask data preparation command that applies a convolution algorithm to input layers to represent density values

Usage

```
DENSITY CONVOLVE input_layer [marker_layer] [ '['density_expr ']'] constraint
[WINDOW wxy]
[INSIDE OF EXTENT | INSIDE OF x1 y1 x2 y2 | INSIDE OF LAYER layer2]
[TRUNCATE | WRAP] [PRINT [ONLY] filename]
FILE {parameter_block | '['
  {{SCALE c_n GAUSS s_n [CONVOLUTION_GRID cg_n]
    [NORMALIZATION type]...} |
  {SCALE c_n FRACTAL {n_n | filename_n} RMAX rmax_n [RMIN rmin_n]
    [CONVOLUTION_GRID {cg_n | AUTO}] [PARTIAL {AUTO | r1_n r2_n}]
    [NORMALIZATION type] [skip]}...}
  [spatialPolynomial [a0] [a1x] [a2y] [a3xy] [a4x2y] [a5xy2] ...]
  [upsample_interpolation [linear | none]]
  [{input_mask format_specific_options} |
  {vboasis_path file_path [-noCheck] [-fastIndex [layer datatype]]}
    [vboasis_precision_multiplier {n[/ d] | AUTO}]
    [vboasis_layout_magnify n [/ d]]
    [vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]}
  [density_algorithm {1 | 2}]
  [direct_FS_access INPUT]
  [dummy_fill {value | auto}]
  [file_format {text | binary | png [gray | grey | color | colour]}]
  [<SVRFSTART>
    svrf_statements
  <SVRFEND> ]
  [svrf_layer_name {layer_name | {layer_name oasis_layer_number
    [oasis_layer_datatype]...}}]
  [density_precision [double | float]]
  [dose_ring dose delta_x_um delta_y_um]
  [blade_shadow blade_x_um blade_y_um]
  [RDB [ONLY] filename [MAG value]]
  [globalMapFile {binary | ascii} file_path]
  [section_size sz]
  }
  ']' }
```

Description

The DENSITY CONVOLVE SVRF statement allows you to create a representation of your input layer that shows density values after applying convolution algorithms. Convolution is a mathematical procedure that produces a weighted moving average of two functions. In this case,

it evaluates the density value of a density window by taking into consideration the values of those windows surrounding it. Therefore, the density values are “spread out” to nearby density windows.

Arguments

- *input_layer*

The name of an original or derived polygon layer. This is the layer containing the data to which Calibre applies the convolution functionality.

This differs from Density in that you can only specify a single layer.

- *marker_layer*

The name of an original or derived polygon layer. This layer should only be specified with the dummy_fill parameter.

- ‘[’ *density_expr* ‘]’

An optional expression enclosed in brackets ([]) that allows customized control over the density computations. The expression may contain numbers, numeric variables, binary operators (*, /, +, -), unary operators (+, -, !, ~), and AREA functions of the input layer of the form, where the parentheses () are required:

```
AREA (input_layer)
```

The expression must not result in strictly negative values because they cannot be checked by a constraint. The AREA function returns values based upon user units of length squared. AREA() returns the area of the data capture window.

For example, to return the value “1 - density” to the convolution operation you would specify:

```
[1 - AREA(layer_name) / AREA()]
```

- *constraint*

A required constraint bounded by the following::

If the density value, after convolution, of a window meets this constraint, the window is output to the derived layer.

If a *density_expr* is provided, the result of the expression is tested against the constraint.

If the ratio is intended as a percent, it should be a number between 0 and 1, where division by 100 has occurred. For example, if you want the constraint to be 20 percent, you would specify “0.20” not “20”.

You cannot specify the constraint “< 0”.

- WINDOW *wxy*

An optional keyword set that specifies the size of the window within which the tool calculates the density and applies the convolution algorithm.

Specifies a square window with a height and width of wxy user units. The string wxy must be a positive real number. If you do not specify this optional keyword set, the default window size is defined by the INSIDE OF boundary, resulting in a single window.

To prevent any WINDOWS from overlapping, the tool STEPs the window by the same value (wxy).

Consult your process engineer for the best size of the density window. Typically, this would be about two orders of magnitude larger than the nominal feature size.

- **INSIDE OF EXTENT | INSIDE OF $x1\ y1\ x2\ y2$ | INSIDE OF LAYER $layer2$**

An optional keyword set that defines a rectangular boundary within which the tool creates the density grid. The choices are:

INSIDE OF EXTENT — Specifies that the rectangular boundary is the extent of *input_layer*.

INSIDE OF $x1\ y1\ x2\ y2$ — Specifies the lower-left and upper-right corners of the rectangular boundary in user units. You must write the coordinates relative to the origin of the Calibre database and enclose any negative coordinates in parentheses (), for example:

```
INSIDE OF (-123) 123 (-12) 456
```

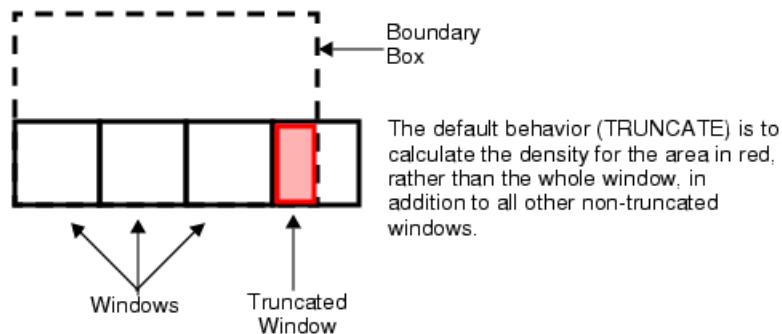
INSIDE OF LAYER $layer2$ — Specifies that the rectangular boundary is the extent of *layer2*. This is different behavior from the Density SVRF statement and *layer2* cannot be the same layer as *layer1*.

- **TRUNCATE | WRAP**

Specifies how the tool calculates the convolution of a window that overlaps the boundary box of the *input_layer*. (Density behavior is always truncate.)

TRUNCATE — An optional keyword that instructs the tool to compute the convolution of the overlapping window by truncating the size of the window so that it does not extend beyond the boundary box. This is the default behavior.

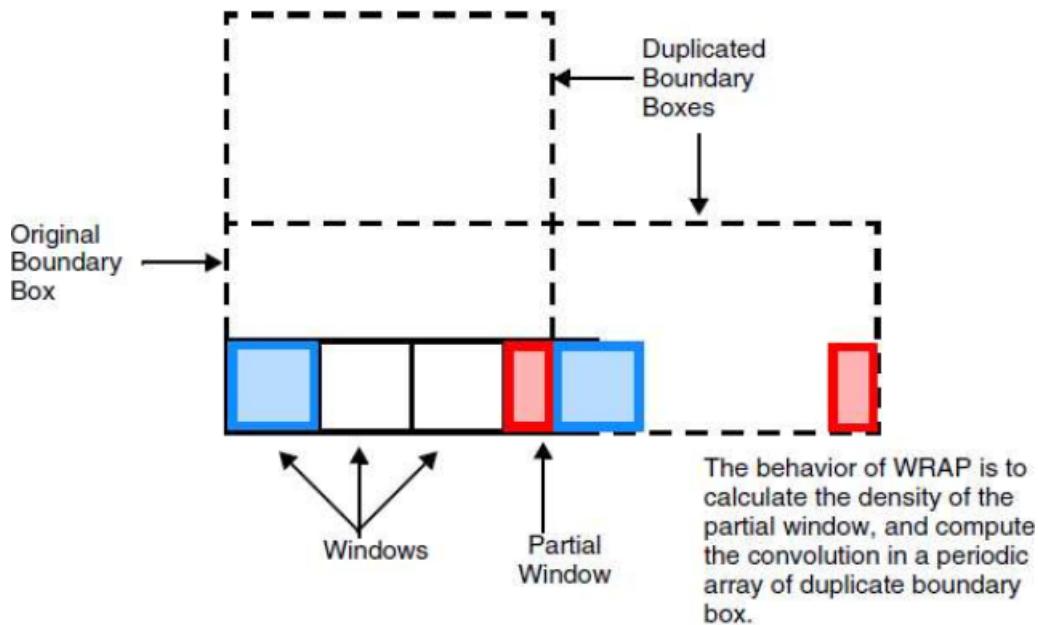
Figure 9-2. TRUNCATE Keyword



WRAP — An optional keyword that instructs the tool to compute the convolution of the overlapping window by duplicating and adding the boundary box and its data to the

right-hand or top side of the main boundary box. The convolution is then taken from the window that intersects the duplicated boundaries.

Figure 9-3. WRAP Keyword



- **PRINT [ONLY] *filename***

An optional keyword set that instructs the tool to write data about each window to the desired location. The choices are:

PRINT *filename*—Writes the window data to the specified *filename* and the result layer.

PRINT ONLY *filename* — Writes the window data only to the specified *filename* and leaves the result layer empty. This saves operation time if you are not interested in the geometric result.

The *filename* parameter can contain environment variables.

The file is a list of all the windows in the following format:

`x1 y1 x2 y2 value`

where,

- `x1 y1 x2 y2` — reports the position of the window
- `value` — reports the density value, after convolution

- **FILE *parameter_block***

A keyword and argument pair specifying the density convolution parameters. The parameters are contained in a reusable parameter block defined using a Litho File SVRF statement.

- **FILE** ‘[’... ‘]’

A keyword and argument set specifying that the density convolution parameters are contained inline within the brackets ([]). Brackets must surround all operation arguments appearing after the keyword FILE. In addition, everything within the brackets must comply with the following requirements:

- Keywords and parameters must be on lines *strictly between* the left and right brackets (that is, cannot be on the same line).
- You can include comments within the section of the operation set off by brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.

- **SCALE** *c_n*

A required keyword and argument pair specifying a scaling factor. This factor is associated with each two-dimensional Gaussian kernel.

- **GAUSS** *s_n*

A required keyword (if you are using Gaussian kernels) and argument pair specifying a Gaussian sigma value. This value represents the sigmas for each two-dimensional Gaussian kernel.

- **FRACTAL** {*n_n* | *filename_n*}

A required keyword (if you are using fractal kernels) and argument pair that specifies a fractal value, representing the power of the two dimensional fractal kernel that will be convolved with the density values. Note that fractal kernels may not be combined with Gaussian kernels, but you may have multiple fractal kernels. The value of *n* must be positive. Alternatively, a filename can be specified that contains a text based fractal kernel that will be read in and used for the convolution. The text file must contain a series of samples that strictly adheres to the following format:

```
// this is a comment
# this is also a comment
// All white space is ignored.
// This is a list of sample values of radius in microns followed by
// the fractal value.
// Only white space is allowed between the radius and fractal value.
<r1> <f1>
<r2> <f2>
...
<rn> <fn>
```

- **RMAX** *rmax_n*

A required argument if you are using fractal kernels that specifies a Fractal maximum radius value in microns, representing the cutoff value for which the Fractal kernel will be set to zero if $r > rmax_n$. The default value is 90 percent of energy in fractal kernel.

- **RMIN** *rmin_n*

An optional argument that specifies a Fractal minimum radius value in microns, representing the cutoff value for which the Fractal kernel will be set to zero if $0 < r < r_{max_n}$. The default value is 1.0 um.

- **CONVOLUTION_GRID** {*cg_n* | **AUTO**}

An optional keyword and argument pair specifying a convolution grid (*cg_n*) for each two-dimensional Gaussian kernel.

You can use this setting to speed up the convolution of the density grid at the expense of precision.

If the **AUTO** keyword is specified for PARTIAL, the **AUTO** keyword must also be specified for CONVOLUTION_GRID.

The default value is the value assigned to GAUSS (*s*) divided by 4.

- **PARTIAL** {**AUTO** | *r1_n r2_n*}

Specifies a partial fractal kernel, where only the portion $r1_n < r_n < r2_n$ of the original fractal kernel is convolved at the given convolution grid. In this manner, the entire original range of the fractal kernel must be built up by using partial kernels. This allows you to customize the convolution grid accuracy and runtime trade-offs associated with very large fractal kernels. It is highly recommended to use “upsample_interpolation linear” when using partial fractal kernels to avoid “stair-stepping” when combining the partial fractal kernels. If **AUTO** is specified, Calibre automatically breaks up the fractal kernel into partial kernels and sets their convolution grids in a reasonable manner to balance both speed and accuracy trade-offs. If **AUTO** is specified, the corresponding **AUTO** setting must be set on the CONVOLUTION_GRID keyword as well.

- **NORMALIZATION** { area | **none** }

Optional normalization flag for each kernel. The “area” option normalizes the convolution kernel to have an area of 1.0, which is then multiplied by the scale factor, while “none” performs no normalization on the convolution kernel but is multiplied by the scale factor. The default setting is area.

- **skip**

This secondary keyword is only used by fractal kernels. It specifies that this particular kernel is supposed to be “skipped” (not performed). This is useful when you would like a particular partial range of the kernel to be handled by the EUV flare simulator.

- **spatialPolynomial** [*a0*] [*a1x*] [*a2y*] [*a3xy*] [*a4x2y*] [*a5xy2*] ...

Defines up to a 4th order polynomial that is evaluated on the user units of the layout that is added to the convolution output. Coefficients can be negative or positive, while the order term must place ‘x’ before ‘y’. Only non-zero coefficients need to be specified.

- **upsample_interpolation [linear | none]**

Specifies that Calibre use linear interpolation on upsampling to reduce “stair-stepping” from mismatched density and convolution grids. By default, upsampling is performed without linear interpolation.

- **input_mask *format_specific_options***

An optional argument for direct input of VSBJOB or MEBESJOB jobdecks or MEBES pattern files for density calculations. When this parameter is specified, Density Convolve uses the geometrical data from the specified mask instead of the input hierarchical layer. All data in the mask is merged. If the extent is specified with INSIDE OF, it must conform to the coordinate system of the jobdeck.

VSBJOB: *jobdeck_path mask_name* — Specify a directory path and mask name. If used with density_algorithm 2, the VSBJOB should have only VSB12 variant chips.

MEBES: *mebes_pattern_path* — Specify a directory path to the MEBES pattern.

MEBESJOB: *jobdeck_path [-levels ALL | list] [-ejobdeck_pattern_suffix *suffix*] [-data_search_path *path*]* — Specify a jobdeck file path and additional options that control what is read in.

GDS placements in extended MEBES jobdecks are not supported when used with density_algorithm 1 or 2. Reverse-tone placements in MEBESJOB are not supported with density_algorithm 2.

-levels ALL | *list* — The levels to consider. By default, all levels are included in the density computation. If using a list, separate levels by commas. Ranges are allowed but not space. For example: “-levels 501-504,508,512”

-ejobdeck_pattern_suffix *suffix* — Allows pattern files with different nomenclature (such as the OASIS .oas extension) to be read in MEBES extended job decks.

-data_search_path *path* — Specifies the path for a data search. Separate directories with a colon (:).

This example specifies a job deck that includes OASIS files located in directories “.”, *mb*, and *om*. All levels are read in.

```
input_mask ./jobdeck/tsisram.om_mb.ejb -ejobdeck_pattern_suffix oas
-data_search_path .:mb:om
```

- **vboasis_path *file_path***

A file path to the optional direct input OASIS file. If this parameter is specified, input geometrical data for the density computation is taken from the specified OASIS file instead of the input hierarchical layer. All data in the file is merged to form the input to the density computation. The specified extent must match that of the OASIS file. Several restrictions and caveats apply to the use of this alternate input method:

- Limited compile-time extent checking is possible since the extent of the OASIS file is not known until it is parsed entirely. The extent is thoroughly checked at run time.
- For distributed processing (Calibre MTflex) with 32-bit remotes, the maximum allowed data size of any cell in the OASIS file is 2GB, although the practical limit

will be determined by the remote memory. Files generated by conversion from any fracture format will always have sufficiently small cells.

- Oasis file indices. Section mode will always attempt to reuse an existing index file for the input VB:OASIS file. Hierarchical mode will always rebuild the index in order to check that the file contains only trapezoids. Any index constructed by the fracture command, whether in hierarchical or section mode, will not contain viewer components, and so may not be suitable for use in the viewer.

While any valid VB:OASIS file (considering the restrictions) can be used as an argument to this keyword, use only OASIS.VSB files (a standardized subset of VB:OASIS) or the outputs of the Calibre MDPview format-to-oasis converters. These files have hierarchies that can be effectively used as spatial indices. Using VB:OASIS files not constrained by this trait will likely result in unacceptably poor fracture performance.

Several side effects occur when input is taken from a VB:OASIS file. The HDB precision is temporarily replaced by the precision in the VB:OASIS file for the duration of the fracture command. Also, the default HDB extent may no longer be a meaningful way of specifying the DENSITY CONVOLVE window, since there is no relationship between the HDB and input OASIS file.

The Density Convolve statement may not specify both vboasis_path and input_mask.

- -noCheck

An optional argument to vboasis_path. The -noCheck option can be specified in circumstances when checks on the input VBOASIS layout file are not possible because it has not been generated. This would typically happen when the file is expected to be generated by another SVRF command and thus is not available during parsing for checking purpose. When using -noCheck, extra care should be given to specifying the file path and associated precision values to avoid errors that will be encountered much later.

- -fastIndex [*layer datatype*]

An optional argument to vboasis_path. The -fastIndex option generates the index more quickly, but is restricted to OASIS.MASK files that contain only a single layer and datatype.

Note



Do not use this option when there are multiple layers or datatypes in the input.

The default *layer* and *datatype* are 0 0.

- vboasis_precision_multiplier {*n* [/ *d*] | AUTO}

When direct input is specified using the vboasis_path keyword, this keyword will multiply the precision of the oasis file before operating on it. The precision will be multiplied by the ratio created by the numerator (*n*) and optional denominator (*d*) integer input. The default for *d* is 1 if otherwise unspecified.

If the AUTO is supplied instead of a ratio, Calibre will attempt to infer the correct ratio but will fail if it is not a rational number, or if the combination of inferred numerator and

denominator would cause arithmetic overflow. This keyword requires the use of embedded SVRF.

- **vboasis_layout_magnify *n* [/ *d*]**

Aligns the OASIS-direct-input coordinate system with the HDB coordinate system by adjusting its scale. The scale is specified as the numerator and denominator of a rational number.

This keyword is primarily used to apply embedded SVRF at the scale of the mask, but after all inverse transforms have been applied to align the MDP file inputs with the HDB coordinate system. In this case one would scale up the HDB coordinate system using LAYOUT MAGNIFY, and also use this keyword to apply the same scale to the direct-input database.

- **vboasis_injection [-size *bin_size* | -size_factor *size_multiplier*] [-preserve 0 | 1]**

The VB:OASIS file specified using vboasis_path may contain cells that have large amounts of data and have a large extent. These large cells degrade performance in section mode processing. To overcome this issue, cells can be partitioned into bins.

The size of the cells that should be binned and the size of each bin (*bin_size*) is determined automatically by default when vboasis_injection parameter is specified. You can control the bin size by providing a *size_multiplier* using the option -size_factor *size_multiplier*. The default value is 1.0.

Alternately the *bin_size* can be explicitly specified using the option -size *bin_size* in microns. Any cell whose extent has width or height more than the *bin_size* is considered for binning.

The vboasis_injection option creates temporary data which may be reused if it is preserved using the -preserve option. The default value is 0.

The temporary data is read or written in a directory named *file_path.lcb* where *file_path* is argument specified to vboasis_path. The location of this directory is same as that of *file_path*. This behavior can be changed by specifying a colon-separated list of directories using the UNIX environment variable MDPIIndexSearchPath. The directories are searched in the order they are listed for reading or writing the temporary directory.

- **density_algorithm {1 | 2}**

An optional argument that specifies the algorithm for computing density for direct-input VSBJOB and MEBESJOB jobdecks and MEBES pattern files. GDS placements in extended MEBES jobdecks are not supported.

1 — This is the default. It indicates to use the full-featured algorithm.

2 — A faster, more limited algorithm for calculating density. It does not affect convolution speed. This algorithm does not merge geometries. It supports multithreading (Calibre MT) but not distributed processing (Calibre MTflex). MEBESJOB jobdecks with reverse-tone placements are not supported.

Note

 Do not use density_algorithm 2 when the input still contains overlapped polygons. Because geometries are not merged in the faster algorithm, the overlapped area is counted multiple times.

- **direct_FS_access INPUT**

An optional keyword that permits remote processes in a Calibre MTflex run to directly access a file system for direct input (input_mask or vboasis_path).

This keyword is supported only for extended MEBES jobdecks that use only OASIS placements.

The default behavior allows remote processes to access the file system through the primary process using the Calibre Remote Protocol. When “direct_FS_access INPUT” is specified, remote processes read direct input files using system calls. Note that “INPUT” is case sensitive.

- **dummy_fill {value | auto}**

An optional keyword that assigns density to the area covered by *marker_layer*. This can be useful when the layout has large cutouts with zero density, as is common with kerf regions.

A *marker_layer* is required when specifying dummy_fill. The layer may contain multiple polygons.

The dummy_fill keyword must include one of the following arguments that specifies the density to assign:

value — A number from 0 to 1, inclusive.

auto — A keyword that uses the average density of *input_layer* outside of *marker_layer*.

Note that the dummy_fill value is added to any density that would normally be computed during convolution for the region covered by the marker layer.

- **file_format {text | binary | png [gray | grey | color | colour]}**

This optional keyword works in tandem with the PRINT [ONLY] keyword to print density convolve output into a file in specified format. Currently it supports text, binary and PNG formats. In addition, you can specify PNG files to be either in gray or color encoding. The default setting is text.

It is often easier to open and check flare values in text format. However, the file sizes in text format is usually an order of magnitude (10x or higher) larger than the next biggest size which is binary format.

The binary format preserves the precision completely and preferred over other modes if precision is of utmost importance.

PNG files preserve most of the accuracy (especially in color mode), but can also be viewed in one of the PNG supported image viewers. PNG files also uses loss-less compression to reduce file sizes. PNG files in color mode can typically be 3 to 4x smaller than the

corresponding files in binary format. PNG files in gray scale are typically smaller (by 2 to 3x) than PNG files in color mode and have less accuracy compared to color mode. Gray scale PNG format is preferred over other formats when file size is of utmost importance.

- <SVRFSTART>

A keyword indicating the beginning of the embedded SVRF statement group. Further information on using embedded SVRF can be found in the *Calibre Mask Data Preparation User's and Reference Manual*.

- *svrf_statements*

Embedded SVRF statements must obey the same rules as SVRF with the additional constraints:

- Lines cannot be longer than 1023 characters. (Calibre truncates any SVRF line to 1023 characters if you exceed this value.)
- One output rule check with a single polygon output is allowed.
- No left- or right-brackets are allowed.

You must specify a single rule check within the embedded SVRF statement group. The output of this rule check is what Calibre uses as the input to the FRACTURE operation, and must only be polygon data, not edge or error data. Refer to the *Calibre Verification User's Manual* for more information about rule checks.

You cannot specify left- or right-brackets within the embedded SVRF statement group. Therefore, any occurrence of brackets must be replaced as follows:

Left bracket ([)	<LB>
Right bracket (])	<RB>

The precision of the embedded SVRF statements must match that of the Calibre rule deck containing the FRACTURE statement.

- <SVRFEND>

A keyword indicating the end of the embedded SVRF statement group.

Note

 The left and right angle brackets (< >) for <SVRFSTART>, <SVRFEND>, <LB>, and <RB> are literal and required. SVRFSTART and SVRFEND must also always be capitalized.

- *svrf_layer_name {layer_name | {layer_name oasis_layer_number [oasis_layer_datatype]}...}*

A keyword that specifies the names (*layer_name*), numbers (*oasis_layer_number*), and optional data types (*oasis_layer_datatype*) of the embedded SVRF layers corresponding to VB:OASIS geometry. For a single layer, the first form is used. For multiple layers, each name and corresponding layer number in the OASIS file must be specified. When

VB:OASIS input is used with HDB input, the precisions of the HDB and VB:OASIS file must match.

The parameters enclosed in braces ($\{layer_name\} | \{layer_name\} oasis_layer_number [oasis_layer_datatype]\})$ can be repeated 0 or more times.

- **density_precision** [double | float]

This keyword instructs Calibre to internally store the density values with double or floating point precision, respectively. Significant memory savings will occur with float. The default setting is double.

- **dose_ring** *dose* *delta_x_um* *delta_y_um*

Optional keyword that adds a ring of constant additional dose around the border of the convolution area given by points that are within *delta_x_um* of the left and right edges, and *delta_y_um* of the top and bottom edge. The additional dose is ADDITIVE in the corners of the simulation, meaning *dose* will be added twice for any points that satisfy both *delta_x_um* and *delta_y_um*. This is useful when modeling Optical Density or Black Border Reflections. The default is all values of zero. The *delta* values are in microns, and the *dose* value is in intensity.

- **blade_shadow** *blade_x_um* *blade_y_um*

Optional keyword that models the partial shadowing of the wafer by masking blades in the litho tool. This performs much like the *dose_ring* keyword, except it provides a linear roll-off of the dose from the value *dose* in the *dose_ring* keyword to zero at the end of the blade shadowing. Hence, this keyword requires that *dose_ring* set a non-zero *dose* value. In this manner, values within $\delta_x < x < (\delta_x + blade_x_um)$ of the right and left borders of the region, will experience a linear roll-off from *dose* to zero. Values within $\delta_y < y < (\delta_y + blade_y_um)$ of the top and bottom borders of the region will experience a linear roll-off from *dose* to zero. This effect is additive for any points satisfying both x and y conditions. The *blade* values are in microns.

- **RDB** [ONLY] *filename* [MAG *value*]

An optional keyword set that instructs the tool to create an ASCII results database (RDB) with geometry and detailed statistics, having the given filename. File names are case-sensitive. Subdirectories in a pathname are created as required.

RDB *filename* — Specifies RDB output to the *filename*. This database is in addition to the usual Calibre nmDRC Results Database and can be loaded into Calibre RVE or Pyxis Layout.

RDB ONLY *filename* — Specifies results are sent to the RDB only. No geometric data is sent to the usual DRC Results Database. Any results sent to the RDB in this mode are not reported in either the run transcript or the DRC Summary Report.

MAG *value* — Specifies the results sent to the RDB *filename* are magnified by the specified *value*. The *value* must be a positive floating point number, which is dimensionless.

- **globalMapFile {binary | ascii} *file_path***

Specifies a user-defined grid of points that exactly matches the grid of the density function. The points in the grid are added to the convolution result, similar to the spatialPolynomial keyword. You must specify either binary or ASCII file format. The ease of use of an ascii file can be weighed against the smaller file size and faster read in of a binary file. Both options are available. Within the global map file, two integers must be written that specify the number of points in the X-direction and the number of points in the Y-direction. These integers are followed by a matrix of floating-point numbers that specify the global map. The global map must be written in column major order (columns rasterized first).

- **section_size *sz***

Sets the desired size of a square section in user units. This can only be used with direct data entry, and overrides the default chosen settings.

Examples

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "../data/ring.oas"
LAYOUT PRIMARY "ring"

PRECISION 1000
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "convolve_ring_output.oas" OASIS PSEUDO
DRC SUMMARY REPORT report.convolve_ring

LAYER orig3 3

LITHO FILE convolve_parameters [
    SCALE 0.2 GAUSS 1
    SCALE 0.8 GAUSS 48
]

convolve_out_1 = DENSITY CONVOLVE 3 < 0.3 TRUNCATE WINDOW 25 FILE
convolve_parameters

convolve_out_2 = DENSITY CONVOLVE 3 >= 0.3 < 0.4 TRUNCATE WINDOW 25 FILE
convolve_parameters

convolve_out_3 = DENSITY CONVOLVE 3 >= 0.4 TRUNCATE WINDOW 25 FILE
convolve_parameters

orig3 {COPY orig3 } DRC CHECK MAP orig3 3
convolve_out_1 {COPY convolve_out_1} DRC CHECK MAP convolve_out_1 301
convolve_out_2 {COPY convolve_out_2} DRC CHECK MAP convolve_out_2 302
convolve_out_3 {COPY convolve_out_3} DRC CHECK MAP convolve_out_3 303
```

Example of Gaussian Kernel Definitions

```
LITHO FILE dx [
    scale 8.2 gauss 100.000000
    scale -6.50 gauss 1000.000000
    vboasis_path input.oas
]
m = density convolve prime0 < 0.200 >= 0.100000 wrap window 100 inside of \
34695.968000 27518.852 36687.824 29565.584 file dx
```

Example of Fractal Kernel Definitions

```
LITHO FILE convolve_param [
    scale 1 fractal 2 rmin 1 rmax 250 convolution_grid 10 partial 160 250
    scale 1 fractal 2 rmin 1 rmax 250 convolution_grid 10 partial 90 160
    scale 1 fractal 2 rmin 1 rmax 250 convolution_grid 10 partial 40 90
    scale 1 fractal 2 rmin 1 rmax 250 convolution_grid 10 partial 1 40
    upsample_interpolation linear
]
d10 = density convolve poly [1-area(poly)/area()] <0.1 window 10 file \
convolve_param
```

Report Example

```
TIME FOR CONVOLUTION PHASE: CPU TIME = 0 REAL TIME = 0 LVHEAP = 1/3/3
TIME FOR OUTPUT PHASE: CPU TIME = 0 REAL TIME = 0 LVHEAP = 1/3/3
TIME FOR EXPRESSION EVALUATION, SUBSAMPLING, CONVOLUTION, OUTPUT PHASE:
CPU TIME = 0 REAL TIME = 0 LVHEAP = 1/3/3
convolve_out_1 (HIER TYP=1 CFG=1 HGC=1 VHC=F VPC=F)
convolve_out_2 (HIER TYP=1 CFG=1 HGC=3 FGC=3 VHC=F VPC=F)
convolve_out_3 (HIER TYP=1 CFG=1 HGC=5 FGC=5 VHC=F VPC=F)
CPU TIME = 0 REAL TIME = 0 LVHEAP = 1/3/3 OPS COMPLETE = 5 OF 8
```

The Hierarchical Geometries Count (HGC) value for each output layer shows you the number of geometries created. These geometries are merged representations of all the density windows that met the constraint for the given Density Convolve calculation.

ASCII Density Map Example

This example reads a full-mask job deck and writes a density map in ASCII format to the file *print.dc00.txt*. The map contains density values for each 100 um x 100 um window. Because PRINT ONLY is specified, the dc00 layer does not contain any density data.

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "oas_in/dummy_1000.oas"
LAYOUT PRIMARY "*"
LAYOUT MONDO FLEX yes

PRECISION 1000

DRC RESULTS DATABASE "./oas_out/jobdeck_density.map.oas" OASIS PSEUDO
DRC MAXIMUM RESULTS ALL
DRC SUMMARY REPORT "./summaryrep/summary.jobdeck_density.map.txt"

LAYER dummy 98
LAYER dataextent 99
```

```
POLYGON 0 0 152400 152400 dataextent

LITHO FILE convolve_parameters [
    SCALE 1.0 GAUSS .0001 CONVOLUTION_GRID 100
    input_mask ./ejb/map.ejb -levels 1
]

dc00 = DENSITY CONVOLVE dummy <= 1.0 WINDOW 100 INSIDE OF LAYER dataextent
      PRINT ONLY print.dc00.txt FILE convolve_parametersdc00 { COPY dc00
} DRC CHECK MAP dc00 OASIS 2000
```

You can derive a single value for the density of the entire mask by averaging the density values of all windows. For this example, you could compute the average with the following awk script in the terminal window:

```
awk '{ total += $5 }; END { print total/NR }' print.dc00.txt
```

Binned OASIS and ASCII Density Maps Example

This example is similar to “[ASCII Density Map Example](#)”, but it separates the values into ten bins. In addition to the ASCII files containing exact density values per window, it writes a graphical version of the density map to an OASIS file (*./oas_out/jobdeck_density.map.oas*).

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "oas_in/dummy_1000.oas"
LAYOUT PRIMARY "*"
LAYOUT MONDO FLEX yes

PRECISION 1000

DRC RESULTS DATABASE "./oas_out/jobdeck_density.map.oas" OASIS PSEUDO
DRC MAXIMUM RESULTS ALL
DRC SUMMARY REPORT "./summaryrep/summary.jobdeck_density.map.txt"

LAYER dummy 98
LAYER dataextent 99

POLYGON 0 0 152400 152400 dataextent

LITHO FILE convolve_parameters [
    SCALE 1.0 GAUSS .0001 CONVOLUTION_GRID 100
    input_mask ./ejb/map.ejb -levels 1
]
```

```

dc01 = DENSITY CONVOLVE dummy      <= 0.1 WINDOW 100 INSIDE OF LAYER
      dataextent PRINT print.dc01.txt FILE convolve_parameters
dc02 = DENSITY CONVOLVE dummy > 0.1 <= 0.2 WINDOW 100 INSIDE OF LAYER
      dataextent PRINT print.dc02.txt FILE convolve_parameters
dc03 = DENSITY CONVOLVE dummy > 0.2 <= 0.3 WINDOW 100 INSIDE OF LAYER
      dataextent PRINT print.dc03.txt FILE convolve_parameters
dc04 = DENSITY CONVOLVE dummy > 0.3 <= 0.4 WINDOW 100 INSIDE OF LAYER
      dataextent PRINT print.dc04.txt FILE convolve_parameters
dc05 = DENSITY CONVOLVE dummy > 0.4 <= 0.5 WINDOW 100 INSIDE OF LAYER
      dataextent PRINT print.dc05.txt FILE convolve_parameters
dc06 = DENSITY CONVOLVE dummy > 0.5 <= 0.6 WINDOW 100 INSIDE OF LAYER
      dataextent PRINT print.dc06.txt FILE convolve_parameters
dc07 = DENSITY CONVOLVE dummy > 0.6 <= 0.7 WINDOW 100 INSIDE OF LAYER
      dataextent PRINT print.dc07.txt FILE convolve_parameters
dc08 = DENSITY CONVOLVE dummy > 0.7 <= 0.8 WINDOW 100 INSIDE OF LAYER
      dataextent PRINT print.dc08.txt FILE convolve_parameters
dc09 = DENSITY CONVOLVE dummy > 0.8 <= 0.9 WINDOW 100 INSIDE OF LAYER
      dataextent PRINT print.dc09.txt FILE convolve_parameters
dc10 = DENSITY CONVOLVE dummy > 0.9 <= 1.0 WINDOW 100 INSIDE OF LAYER
      dataextent PRINT print.dc10.txt FILE convolve_parameters

dc01 { COPY dc01 } DRC CHECK MAP dc01 OASIS 2001
dc02 { COPY dc02 } DRC CHECK MAP dc02 OASIS 2002
dc03 { COPY dc03 } DRC CHECK MAP dc03 OASIS 2003
dc04 { COPY dc04 } DRC CHECK MAP dc04 OASIS 2004
dc05 { COPY dc05 } DRC CHECK MAP dc05 OASIS 2005
dc06 { COPY dc06 } DRC CHECK MAP dc06 OASIS 2006
dc07 { COPY dc07 } DRC CHECK MAP dc07 OASIS 2007
dc08 { COPY dc08 } DRC CHECK MAP dc08 OASIS 2008
dc09 { COPY dc09 } DRC CHECK MAP dc09 OASIS 2009
dc10 { COPY dc10 } DRC CHECK MAP dc10 OASIS 2010

```

Direct Input and Output

When running Density Convolve such that it reads directly from the job deck (for example, using the input_mask or vboasis_path arguments) and writes output to a text file or color map bitmap, the Calibre parser still requires the typical input and output statements. These examples show some “dummy” statements that satisfy the parser. These examples do not show specific Density Convolve values.

One common method directs the RDB output to */dev/null*. (Recall, Density Convolve is set up to write directly to a file other than the RDB.)

```

LAYOUT PATH "empty.oas"
LAYOUT SYSTEM OASIS
LAYOUT PRIMARY "*"

DRC RESULTS DATABASE "/dev/null" ASCII
DRC MAXIMUM RESULTS ALL

```

Another method uses Layout Place Cell instead of DRC Results Database. The following lines create a file named *./dummy.oas* with a precision of 10000. This technique is preferable if combining Density Convolve with other SVRF that produce output.

DENSITY CONVOLVE

```
LAYOUT PATH      "dummy.oas"
LAYOUT PRIMARY   "*"
LAYOUT SYSTEM    OASIS

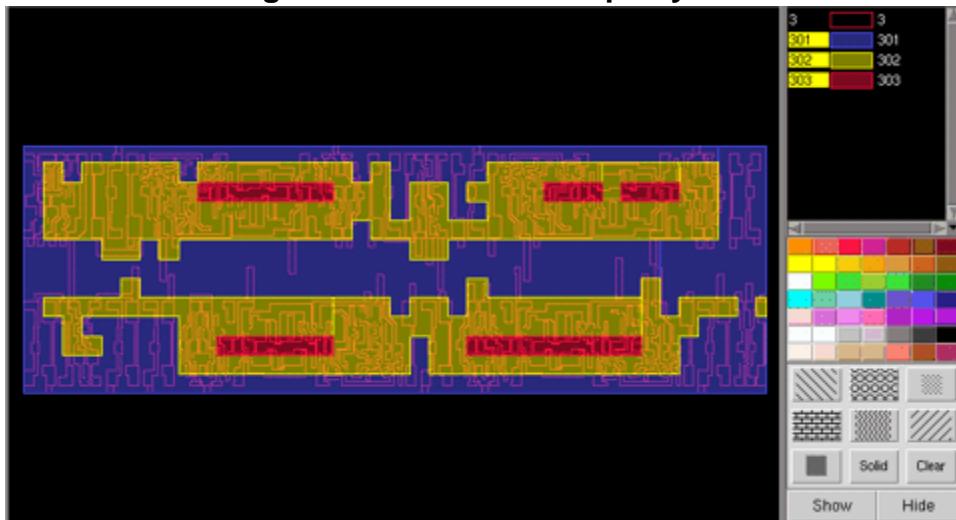
LAYOUT PLACE CELL "dummy.oas" OASIS TOP EMPTY
PRECISION     10000
```

MDP MAPSIZE SVRF Statement

The MDP MAPSIZE SVRF statement allows you to variably resize elements of your layout based on their geographical locations, which are determined by map-size regions. The sizing values are selected as a response to the strength of a physical process effect that requires a data-based correction. Examples are density-based process-loading effects and variation of processing by radial location within a particular machine.

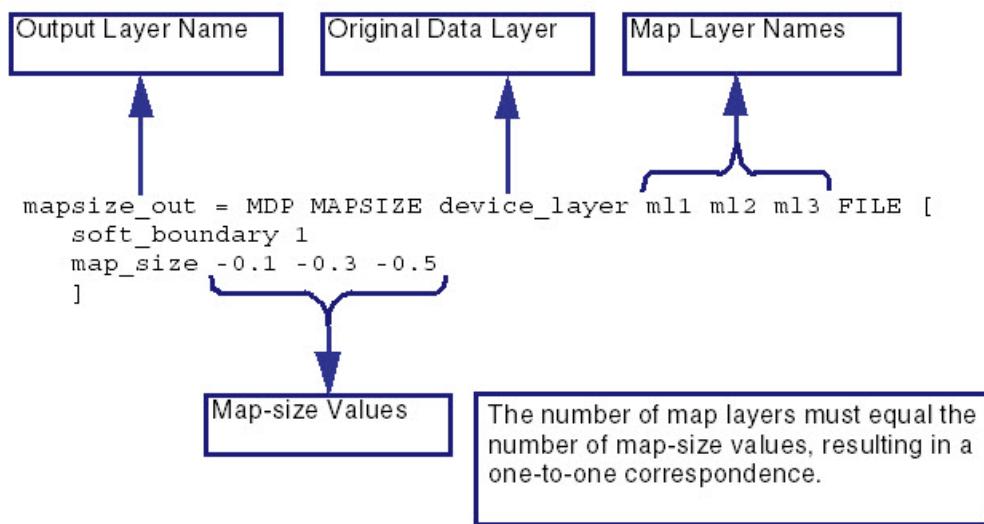
To prepare your data for map-sizing, you must partition your data into non-overlapping map-size regions by creating additional layers in your design data. These new layers are referred to as map layers. You can create these layers either manually, by drawing marker shapes in a layout editor like Calibre MDPview, or derive them inside Calibre using DENSITY, DENSITY CONVOLVE, or other SVRF operations. [Figure 9-4](#) shows an example of what a design could look like with map-size regions. The graphics and examples for this section are based on the output of the DENSITY CONVOLVE examples.

Figure 9-4. Data With Map Layers



You can resize elements within each map-size region by a different value. [Figure 9-5](#) shows an example of a MDP Mapsize SVRF statement, and the full syntax is available in section “[MDP MAPSIZE](#)” on page 270.

Figure 9-5. MDP Mapsize SVRF Example



The output layer (`mapsize_out`) contains the resized data, with the map-size values applied to the edges of the elements of the original layer in accordance to their location relative to the marker layers (map-size regions).

When Calibre resizes the polygons on the original data layer (*input_layer*) it actually resizes the individual edges of the polygons, which means that any polygon that crosses a map-size region could have edges sized by different values and in different directions, depending on whether the map-size value is positive or negative.

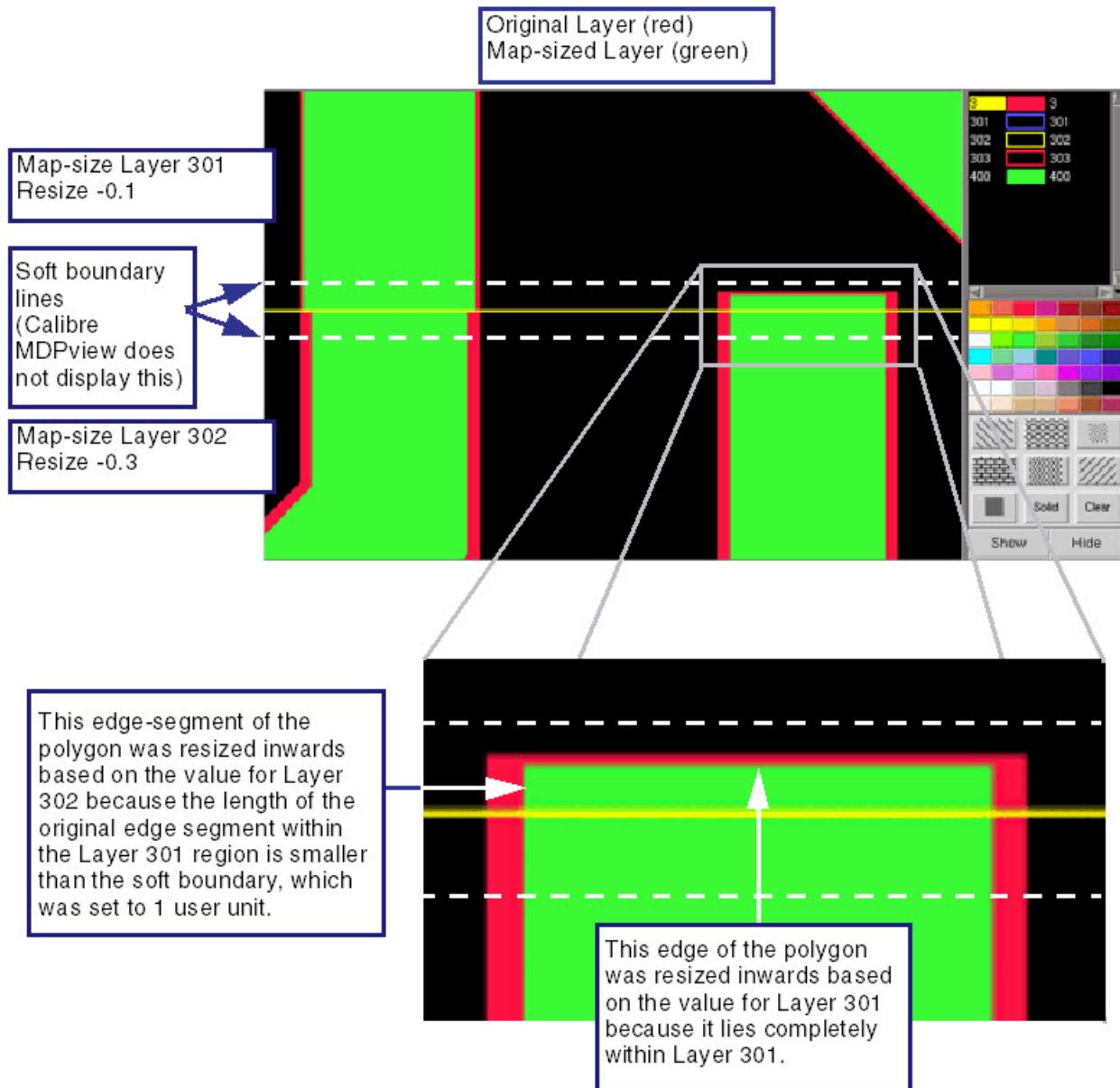
In order to avoid unnecessary jogs in polygons on the output layer, which allows for better lithographic performance and maintains the best possible data quality on the output of the subsequent fracture step (shot count reduction), Calibre sets the exact cutline between adjacent map-size regions within a margin around the region border. You specify this margin size with the **soft_boundary** keyword. Any edge that is within the soft boundary is resized by the value of the region containing the majority of the edge. If a shape crosses the soft boundary on either side of the region border, the transition between the variable sizing values for each region is aligned with an existing jog in the pattern. Only when there are no existing vertices in the entire `soft_boundary` area will the polygon be cut along the border.

After breaking, an edge fragment is assigned to the map region than contains its longest segment. This effectively “stretches” the sharp boundary of a map region to contain a little more. The parameters for this operation define this threshold as well as the sizing values. In the case of an edge lying on the boundary between two regions, the bias applied is that of the map region specified earliest in the map layer list.

[Figure 9-6](#) shows an example of how Calibre resizes geometries that cross region boundaries and soft boundaries.

Operations such as map-size can severely degrade the hierarchy of the input layer. For this reason, map-sizing is frequently used as an embedded SVRF command in section-processing mode.

Figure 9-6. Soft Boundary Behavior



MDP MAPSIZE **270**

MDP MAPSIZE

SVRF mask data preparation command to resize layout elements based on geographical locations

Usage

```
MDP MAPSIZE input_layer map_layer_1 [... map_layer_N]  
{FILE parameter_block | FILE '['  
    soft_boundary {0 | sb_value}  
    map_size {size_value_1 [... size_value_N]}  
    ']'  
}
```

Description

The MDP MAPSIZE SVRF statement allows you to variably resize elements of your layout based on their geographical locations, which are determined by map-size regions. The sizing values are selected as a response to the strength of a physical process effect that requires a data-based correction.

Arguments

- ***input_layer***

A required keyword that specifies the name of an original or derived polygon layer. This is the layer that contains the data to which Calibre applies the map-sizing functionality.

- ***map_layer_1* [... *map_layer_N*]**

A required keyword that specifies the name(s) of an original or derived polygon layer. You can specify this argument any number of times. These layers specify your map-size regions. The number of *map_layer* arguments must equal the number of *size_value* arguments to the **map_size** keyword.

- **FILE *parameter_block***

A required keyword that specifies that the map size parameters are contained in a reusable parameter block defined using the **LITHO FILE** SVRF statement.

- **FILE '[' ... ']'**

A required keyword that specifies that the map size parameters are contained inline within the square brackets ([]). Square brackets must surround all parameters appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:

- Keywords and parameters must be on lines strictly between the left and right brackets (that is, they cannot be on the same line).
- You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.

- **soft_boundary {0 | sb_value}**

A required keyword that specifies the minimum size, in user units, for which Calibre creates an edge segment for edges that cross region boundaries. The default value is 0.

- **map_size size_value_1 [... size_value_N]**

A required keyword that instructs Calibre to perform a map-sizing operation on the data from *input_layer*, where *size_value* is the map-size value, in user units, for the respective *map_layer*. The number of *size_value* arguments must match the number of *map_layer* arguments.

Examples

The following is a complete rule file used for resizing the original layer (layer 3).

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "convolve_ring_output.oas"
LAYOUT PRIMARY "ring"

PRECISION 1000
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "ring_mapsize_output.oas" OASIS PSEUDO

LAYER orig3 3
LAYER orig301 301
LAYER orig302 302
LAYER orig303 303

mapsize_out = MDP MAPSIZE 3 301 302 303 FILE [
soft_boundary 1
map_size -0.1 -0.3 -0.5
]

orig3 {COPY orig3 } DRC CHECK MAP orig3 3
orig301 {COPY orig301 } DRC CHECK MAP orig301 301
orig302 {COPY orig302 } DRC CHECK MAP orig302 302
orig303 {COPY orig303 } DRC CHECK MAP orig303 303
mapsize_out {COPY mapsize_out} DRC CHECK MAP mapsize_out 400
```

MDP CHECKMAP SVRF Statement

The MDP CHECKMAP SVRF statement is used primarily for OASIS bin injection. The command outputs single or multiple layers and datatypes with bin injection for large cells. The criteria for bin injection are based on the physical extent and geometry content of a cell.

See “[Indexing and Injection for OASIS Files](#)” on page 313 for best practices relating to indexing and injection.

MDP CHECKMAP [273](#)

MDP CHECKMAP

SVRF mask data preparation command used output layers and data types with bin injection for large cells

Usage

```
MDP CHECKMAP input_layer
[FILENAME name_of_file] FILE {parameter_block | '['
    [section_size sec_value]
    [output_layer [layer | layer datatype]]
    [cblock]
    [prefix string] [append string]
    [strict {0 | 1}]
    ']'
}
```

Description

The MDP CHECKMAP SVRF statement is used primarily for OASIS bin injection. The command outputs single or multiple layers and datatypes with bin injection for large cells.

Arguments

- ***input_layer***
A required keyword that specifies the name of an original or derived polygon layer. This is the layer that contains the data to which Calibre applies the map-sizing functionality.
- **FILENAME *name_of_file***
An optional keyword that specifies the name of the file.
- **FILE *parameter_block***
A required keyword that specifies that the parameters are contained in a reusable parameter block defined using the [LITHO FILE](#) SVRF statement.
- **FILE '[' ... ']**
A required keyword and argument set specifying that the map size parameters are contained inline within the square brackets ([]). Square brackets must surround all parameters appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:
 - Keywords and parameters must be on lines strictly between the left and right brackets (that is, they cannot be on the same line).
 - You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.

- **section_size *sec_value***
An optional keyword that specifies the size of the square section in microns for large cells. The default value is 1000 um.
- **output_layer [*layer* | *layer datatype*]**
An optional keyword that specifies the output *layer* or *layer* and *datatype* pair mapping. The number of the *layer datatype* pairs must match the number of input layers.
- **cblock**
An optional keyword that specifies that the output is compressed using the CBLOCK record as defined by the OASIS format. Compression is done only when it gives an advantage in terms of amount of data.
- **prefix *string***
An optional keyword that instructs Calibre to prefix all cell names with *string*. This keyword is only valid for GDSII or OASIS format databases. This behavior applies only to results from Calibre nmDRC-H.
- **append *string***
An optional keyword that instructs Calibre to append all cell names with *string*. This keyword is only valid for GDSII or OASIS format databases. This behavior applies only to results from Calibre nmDRC-H.
- **strict {0 | 1}**
An optional keyword that specifies the “strict” OASIS reading mode. The OASIS format has a strict mode that can be read more efficiently. This optional parameter controls whether or not the OASIS output layout file is strict mode or not. If strict is set to 1, the output is in strict mode. If set to 0, the output is not in strict mode. The default is 1.

Examples

Example1

```
mapped = MDP CHECKMAP nw pa p1 pp na pd m2 cc m1 v1 p1_opc
        FILENAME "./flat_mapped_datatypes.oas" FILE [ \
        section_size 1000
        output_layer 1 2 6 7 8 ( 11 3 ) ( 12 4 ) ( 13 5 ) ( 14 6 ) \
        ( 15 7 ) ( 28 8 )
    ]
mapped { COPY mapped } DRC CHECK MAP mapped 99
```

Example 2

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "input/tsisram_flat.oas"
LAYOUT PRIMARY "TOP"
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "results/junk.oas" OASIS PSEUDO LAYOUT ULTRA FLEX YES
LAYER p1_opc 28
mapped = MDP CHECKMAP p1_opc FILENAME "oas_results/
tsisram_checkmap_flat_ultra.oas" FILE [
    output_layer 28
    section_size 10
    PREFIX Hello
    APPEND World
]
mapped { COPY mapped } DRC CHECK MAP mapped 99
```

MDP OASIS_EXTENT SVRF Statement

The MDP OASIS_EXTENT SVRF operation computes the extent of an OASIS layout. The extent can be calculated for a subset of the layout defined in terms of a combination of layer and datatype numbers called layout_subset. Each extent is written to the desired output layer as a rectangle.

In the case of multiple layout_subset parameters, there should always be a matching layer name in the MAP statement.

For all MDP syntax, text must be on lines strictly between the left and right brackets (in other words, they cannot be on the same line). The text should not contain a right bracket (]) or left bracket ([).

MDP OASIS_EXTENT..... [277](#)

MDP_OASIS_EXTENT

SVRF mask data preparation command that computes the extent of an OASIS layout

Usage

```
MDP_OASIS_EXTENT [MAP_ID]
FILE {parameter_block} | '['
[layout_subset name [layer_range datatype_range] ...]
[vboasis_path file_path [-noCheck]]
[vboasis_precision_multiplier {n/d} | AUTO}]
']'
}
```

Description

The MDP_OASIS_EXTENT SVRF operation computes the extent of an OASIS layout. The extent can be calculated for a subset of the layout defined in terms of a combination of layer and datatype numbers called layout_subset. Each extent is written to the desired output layer as a rectangle.

Arguments

- **MAP_ID**

An optional keyword that specifies the MAP identifier. Multiple extent outputs are assigned to a particular layout_subset using the MAP identifier. The extent of the specified layout_subset is the output for that layout_subset. Map identifiers must have a one-to-one, complete mapping to the specified extent scope. Only polygon-directed rule output layers are allowed.

- **FILE parameter_block**

A required keyword that specifies that the parameters are contained in a reusable parameter block defined using the **LITHO FILE** SVRF statement.

- **FILE '[' ... ']**

A required keyword that specifies that the map size parameters are contained inline within the square brackets ([]). Square brackets must surround all parameters appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:

- Keywords and parameters must be on lines strictly between the left and right brackets (that is, they cannot be on the same line).
- You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.

- **layout_subset** *name* [*layer_range datatype_range*]...

An optional keyword that defines the subset of layout whose extent is required. Multiple layout_subset statements can be specified, each identified by a unique name that is used to identify this layout subset in the MAP identifier. If MAP is not specified, only one layout_subset can be used. The layout_subset defines the subset of the input layout using one or more combinations of layer and datatype numbers. The *layer_range* can be either a number, a range of numbers, or an asterisk (*) for all layers. Similarly, *datatype_range* is also a number, a range of numbers, or an asterisk (*) for all data types. The range of numbers is specified using a hyphen between two numbers.

- **vboasis_path** *file_path* [-noCheck]

An optional keyword that specifies the path of the input OASIS file from which extents need to be computed. If an index file exists, then it can be recreated to add the extent information. Several restrictions and caveats apply to the use of this alternate input method, documented in the Direct Oasis Input arguments description in “[Global Calibre FRACTURE Arguments](#)” on page 29.

- **vboasis_precision_multiplier** {*n[/d*} | AUTO}

An optional keyword that multiplies the precision by the supplied number *n*, and then scales the data from the file by the same amount to retain the same absolute scale as the original precision. The default value for *n* is 1. The */d* is used if you wish to express a rational number as a fraction (for example, a multiplier of 1.5x would be declared as “3/2”).

There may be occasions where you might wish to re-interpret the precision of the OASIS file specified using the vboasis_path keyword. For instance, you may want a higher resolution to allow finer shape modifications in embedded SVRF code.

If AUTO is supplied instead of a ratio, Calibre attempts to infer the correct ratio but fails if it is not a rational number, or if the combination of inferred numerator and denominator would cause arithmetic overflow. This keyword requires the use of embedded SVRF.

Examples

The following is a basic example of MDP_OASIS_EXTENT:

```
LAYER dummy 0

LITHO FILE dummy [
    vboasis_path "oas_in/tsisram_1000.oas"
    layout_subset 128_0 28 0           //layer28 datatype 0
    layout_subset 128_0_3 28 0-3      //layer28 datatypes 0-3
    layout_subset 128 28 *            //layer28
    layout_subset all * *             //all layers all datatypes
    layout_subset 11_6 1-6 *          //layers 1-6 all datatypes
]

128_0 { MDP_OASIS_EXTENT FILE dummy MAP 128_0 } DRC CHECK MAP 128_0 12
128_0_3 { MDP_OASIS_EXTENT FILE dummy MAP 128_0_3 } \
DRC CHECK MAP 128_0_3 13
128 { MDP_OASIS_EXTENT FILE dummy MAP 128 } DRC CHECK MAP 128 10
all { MDP_OASIS_EXTENT FILE dummy MAP all } DRC CHECK MAP all 19
11_6 { MDP_OASIS_EXTENT FILE dummy MAP 11_6 } DRC CHECK MAP 11_6 20
```

The following is an example of a fracture operation with an MDP_OASIS_EXTENT-derived fracture extent:

```
LAYER dummy 0

LITHO FILE dummy [
    vboasis_path "oas_in/tsisram_1000.oas"
    layout_subset all * *
]

dataextent = MDP_OASIS_EXTENT
FILE dummy MAP all

frac {FRACTURE OASIS_MASK dummy
    INSIDE_OF LAYER dataextent
    FILE [
        vboasis_path "./oas_in/tsisram_1000.oas"
        file_name "./oas_out/tsisram_extent_vb.oas"
        fracture_units 1000
        computation_mode section
        magnify 4
        svrf_layer_name poly 28
        <SVRFSTART>
            to_fracture {copy poly}
        <SVRFEND>
    ]
}
```

Compare this MDP_OASIS_EXTENT-derived fracture extent definition with one that is explicitly defined in a fracture operation (using INSIDE OF):

```
LAYER dummy 0

frac {FRACTURE OASIS_MASK dummy
      INSIDE OF (-523.500) (-556.500) 523.500 556.500
      FILE [
        vboasis_path "./oas_in/tsisram_1000.oas"
        file_name "./oas_out/tsisram_vb.oas"
        fracture_units 1000
        computation_mode section
        magnify 4
        svrf_layer_name poly 28
        <SVRFSTART>
          to_fracture {copy poly}
        <SVRFEND>
      ]
}
```

MDP DATAPREP SVRF Statement

MDP DATAPREP is an SVRF command used to generate input for data-to-database detection software.

The MDP DATAPREP command requires a separate license (listed as Calibre® D2DBDataPrep) to run. Refer to the *Calibre Administrator's Guide* for further information.

MDP DATAPREP **282**

MDP DATAPREP

SVRF mask data preparation command that generates input for data-to-database detection software.

Usage

```
MDP DATAPREP FILE {parameter_block | '['  
    [section_size size]  
    [output_cell_size size]  
    [care_region x1 y1 x2 y2]  
    [computation_mode {flat | hier}]  
    [log_level lev]  
    [progress_report_interval interval]  
    jobdeck{1|2|3} [-frameChip frame_chip_path]  
    level{1|2|3} level_name level_num  
    output_path pathname  
    mask_type {BINARY | EPSM1 | EPSM2 | CPL | APSM | EAPSM}  
    ']'  
}
```

Arguments

- **FILE parameter_block**

A required argument that specifies that the parameters are contained in a reusable parameter block defined using the [LITHO FILE](#) SVRF statement.

- **FILE '[' ... ']'**

A required keyword that specifies that the map size parameters are contained inline within the square brackets ([]). Square brackets must surround all parameters appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:

- Keywords and parameters must be on lines strictly between the left and right brackets (that is, they cannot be on the same line).
- You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.
- **section_size size**

An optional argument that specifies the section size in microns the tool uses to read the data and process operations. The section_size must be larger and a multiple of the output_cell_size. If not, the tool issues a warning and adjusts the section_size to be a multiple of output_cell_size before continuing. The default value is 250 microns.

- **output_cell_size *size***

An optional argument that specifies the output cell size in microns. Tool will create the output cells with the size specified with this argument. The default value is 50 microns.

- **care_region *x1 y1 x2 y2***

An optional argument that specifies the coordinates of a region that is to be processed. The tool creates output for the region specified with this argument. The default coordinates are [0 0 152400 152400].

- **computation_mode {flat | hier}**

An optional argument that specifies one of the following computation modes:

flat — Specifies flat processing. This is the default setting.

heir — Specifies hierarchical processing. The tool first checks if the following conditions are met:

- The frame chip (as specified in the jobdeck{1|2|3} argument) is a chip that covers all other main patterns of the job deck. The data in frame chip should not intersect the bounding boxes of non-frame chips.
- The bounding boxes of chips other than the frame chip should not intersect one another.
- The properties of chips such as placement bounding box, mirror, tone, and precision of chips that overlap each other should be the same across all job decks.

If these conditions are met, then hierarchical processing will continue. If any are not met, the tool switches to flat processing.

- **log_level *lev***

An optional argument that specifies the log level. The tool issues log messages depending on the level specified (1 for level1, 2 for level2, and 3 for level3). The default value is 1.

- **progress_report_interval *interval***

An optional argument that specifies the interval time in seconds for the progress meter. The default value is 60 seconds.

- **jobdeck{1|2|3} [-frameChip *frame_chip_path*]**

A required argument set that specifies the path to the input job decks.

-frameChip *frame_chip_path* — Optionally specifies a frame chip that covers all other main pattern chips in the job deck. It is assumed that the data in the frame chip does not overlap or intersect the bounding box regions of non-frame chips. If this condition is not met, the tool switches to flat processing.

The number of input job decks required depend on the mask type as defined by the mask_type argument. Each input job deck also has a corresponding layer level, as defined by the level{1|2|3} argument. The following summarizes the relationship between mask type, input job deck, and level.

Table 9-1. Mask Type and Inputs

Mask Type	Job Deck Input	Layer Input
BINARY	jobdeck1	level1
EPSM1	jobdeck1	level1
	jobdeck2	level2
EPSM2	jobdeck1	level1
	jobdeck2	level2
CPL	jobdeck1	level1
	jobdeck2	level2
APSM	jobdeck1	level1
	jobdeck2	level2
EAPSM	jobdeck1	level1
	jobdeck2	level2
	jobdeck3	level3

- **level{1|2|3} level_name level_num**

A required argument set that specifies the level name and level number that the tool must process from their corresponding input job decks. The number of input layers and job decks specified depends on the mask_type specified. [Table 9-1](#) summarizes the relationship between mask type, job deck, and layer levels.

- **output_path pathname**

A required argument that specifies the location where the job deck structure is generated. The tool creates a list of OASIS files along with the job deck containing the placements of the OASIS files and places them in the specified location. The contents in this directory are overwritten if the directory already exists.

- **mask_type {BINARY | EPSM1 | EPSM2 | CPL | APSM | EAPSM}**

A required argument that specifies the mask type. The tool will switch to the corresponding job deck and layer level input rules for the mask (summarized in [Table 9-1](#)), and generates output. BINARY requires one input, EAPSM requires three inputs, and masks EPSM1, EPSM2, CPL, and APSM require two inputs.

Description

The MDP DATAPREP command is used to generate the input for data-to-database detection tools. This command accepts multiple extended MEBES job decks as inputs depending on the type of the mask and executes rules corresponding to the mask, creating an extended MEBES job deck with OASIS placements as output. The input job deck should have only OASIS placements. The database-to-detection software can directly use the output generated by MDP DATAPREP.

Examples

Example 1

The following is an example of an MDP DATAPREP rule block.

```
MDP DATAPREP FILE [
    section_size 250
    output_cell_size 50
    care_region 0 0 152400 152400
    log_level 3

    jobdeck1 job1.ejb -framechip framechip1.oas
    level1 L1 1

    jobdeck2 job2.ejb -framechip framechip2.oas
    level2 L2 1
    output_path output_dir
    mask_type EPSM1
]
```


Chapter 10

Calibre MASKOPT

Calibre® MASKOPT™ is a licensed utility that modifies the input geometry to improve fracture quality. Constrained jog movements are applied to form polygons which, when fractured, have fewer shot counts and fewer small figures than the initial fractured polygons.

The Calibre MASKOPT Setup File	287
Calibre MASKOPT Commands.....	290
LITHO MASKOPT	291
setlayer mdp_maskopt	294
direct_input.....	296
direct_output.....	298

The Calibre MASKOPT Setup File

To use Calibre MASKOPT, create a Calibre MASKOPT setup file (a LITHO FILE command block) to be included as part of your Calibre MDP rule deck.

This setup file (based on a Calibre® OPCverify™ setup file) contains all of the information necessary for running the Calibre MASKOPT operation, including:

- Input layer definitions (including Fracture extent)
- An SVRF LITHO FILE statement to create the setup file
- A direct_input block to input layers directly from a VBOASIS design file
- A setlayer mdp_maskopt definition to apply the Calibre MASKOPT corrections to the input design layers
- A direct_output block to output layers to an OASIS file
- Output layer definitions using LITHO MASKOPT
- The Calibre OPCverify command processing_mode must be set to flat

For more information on the Calibre OPCverify setup files and their commands, refer to the *Calibre OPCverify User's and Reference Manual*.

The following example Calibre MASKOPT setup file illustrates the minimum requirements:

```

LAYOUT SYSTEM OASIS
LAYOUT PATH "dummy.oas"
LAYOUT PRIMARY "*"
PRECISION 1000
LAYOUT ULTRA FLEX YES 128
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "dummy_out.oas" OASIS PSEUDO

//Define input layers, including the fracture extent layer.
LAYER dummy1 1
LAYER fracture_extent 8191

// The POLYGON command specifies the fracture_extent area.
POLYGON 3500 1000 4500 2000 fracture_extent
POLYGON -4305 -4489 10736 9791 fracture_extent
LAYOUT PLACE CELL "dummy.oas" OASIS TOPCELL EMPTY

// Begin the setup file (called "maskopt.setup" with an SVRF LITHO
// FILE statement.
LITHO FILE "maskopt.setup" [
// It is recommended that you set the tilemicrons command
// to 100 (this is the default).
    tilemicrons 100
    processing_mode flat

// Use the direct_input command block to input layers
// directly from a VBOASIS source design. This avoids HDB construction
// as Calibre reads directly from the disk.
    direct_input {
        vboasis_path test_input.oas
        input 1 layer 50 215
        input 2 layer 50 216
    }
// The opc_input and opc_fill are the output from the direct_input block.
    layer m1opc
    layer m1fill

// The setlayer statement performs pre-processing on the output
// (Boolean OR operation)
    setlayer input_final = or m1opc m1fill

// Use the setlayer mdp_maskopt command to apply the Calibre MASKOPT
// operation to improve fracture results.
    setlayer mopt = mdp_maskopt input_final -max_jog_width 0.010 \
        -max_jog_alignment 0.015 -min_edge_separation 0.035 \
        -max_jog_area 0.000075

// Use the direct_output command to output layers directly to an OASIS
// file.
    direct_output {
        vboasis_path out_data.oas
        output input_final layer 0
        output mopt layer 1
    }
]

```

```
input_final = LITHO MASKOPT fracture_extent dummy1 MAP input_final \
    FILE "maskopt.setup"
input_final {COPY input_final} DRC CHECK MAP input_final 0

// Output the results to the mopt layer using LITHO MASKOPT.
mopt = LITHO MASKOPT fracture_extent dummy1 MAP mopt FILE "maskopt.setup"
mopt {COPY mopt} DRC CHECK MAP mopt 1

// The fracture_extent layer is used as the extent in
// to operate on the direct_input layer.
vsb12_frac { FRACTURE NUFLARE mopt INSIDE OF LAYER fracture_extent FILE [
    version 12_v2
    vboasis_path out_data.oas -noCheck
    chip_directory "MASKOPT_VSB12_fracture"
    computation_mode section
    small_value 0.05
    cd internal auto
    conversion_address_unit .001
    max_skew_approximation_error 0.005
    shot_size 0.8
    magnify 4
    frame_width 128
    frame_max_data 2048
    shot_count_stats
]
```

Calibre MASKOPT Commands

The following table lists the required Calibre MASKOPT setup file commands.

Table 10-1. Calibre MASKOPT Commands

Command	Description
LITHO MASKOPT	Defines a LITHO batch operation.
setlayer mdp_maskopt	Modifies the input geometry to improve fracture quality.
direct_input	Enables Direct Data Entry (DDE) for Calibre MASKOPT.
direct_output	Enables direct data output of OASIS files. Used with litho-flat configurations only.

LITHO MASKOPT

SVRF mask data preparation command that defines a LITHO batch operation

Usage

```
output_layer_name = LITHO MASKOPT input_layer_name FILE parameter_block_name
MAP output_layer
```

Arguments

- ***output_layer_name***

A required keyword specifying the name of the new output target layer. The new layer is created and the results of the operation are written to the layer.

- ***input_layer_name***

A required keyword that specifies the input layer(s). The layer specified must be a valid Calibre layer containing polygon data.

- **FILE *parameter_block_name***

A required keyword that specifies the name of a parameter block defined within the SVRF rule file, using the **LITHO FILE** command block. A separate setup file can also be specified.

- **MAP *output_layer***

A required keyword followed by the associated argument indicating the layer containing the output to be returned.

Description

The LITHO MASKOPT command is a tool argument to the Calibre LITHO batch operation. This command is used in conjunction with the **setlayer mdp_maskopt** command to derive output target layers for SVRF.

Examples

The following is a basic example of a call to LITHO MASKOPT.

```
orig_input = LITHO MASKOPT fracture_extent MAP orig_input \
FILE "maskopt.setup"
```

The following is a complete example of using LITHO MASKOPT in a rule file:

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "dummy.oas"=
LAYOUT PRIMARY "*"
LAYOUT PLACE CELL "dummy.oas" OASIS TOP EMPTY

DRC RESULTS DATABASE "maskopt_dummy.oas" OASIS STRICT CBLOCK
DRC MAXIMUM RESULTS ALL
PRECISION 10000
LAYOUT ULTRA FLEX YES

LAYER db_extent 10000
POLYGON 160300 120300 160500 120500 db_extent

dummy_output { AREA (EXTENT) == 0 }
el1 = AREA db_extent == 0
el2 = AREA el1 == 0

LITHO FILE mopts /*

tilemicrons 100
processing_mode flat

direct_input {
    vboasis_path "m1_mpc.oas"
    input 1 layer 515 40100
    input 2 layer 515 41100
    input 3 layer 515 42140
}

layer mpc0
layer mpc1
layer mpc2

set mjw 0.0800
set mjal 0.010
set mes 0.0800
set mjar 0.0004

setlayer mpc0_opt = mdp_maskopt mpc0 \
    -max_jog_width      $mjw \
    -max_jog_alignment   $mjal \
    -min_edge_separation $mes \
    -max_jog_area        $mjar

setlayer mpc1_opt = mdp_maskopt mpc1 \
    -max_jog_width      $mjw \
    -max_jog_alignment   $mjal \
    -min_edge_separation $mes \
    -max_jog_area        $mjar

setlayer mpc2_opt = mdp_maskopt mpc2 \
    -max_jog_width      $mjw \
    -max_jog_alignment   $mjal \
    -min_edge_separation $mes \
    -max_jog_area        $mjar
```

```
direct_output {
    vboasis_path "output/maskopt_out.oas"
    output mpc0_opt layer 515 40100
    output mpc1_opt layer 515 41100
    output mpc2_opt layer 515 42140
}

*/]

mpc0_opt = LITHO MASKOPT db_extent el1 el2 MAP mpc0_opt FILE mopts
mpc1_opt = LITHO MASKOPT db_extent el1 el2 MAP mpc1_opt FILE mopts
mpc2_opt = LITHO MASKOPT db_extent el1 el2 MAP mpc2_opt FILE mopts

mpc0_opt {
    COPY mpc0_opt
}
DRC CHECK MAP mpc0_opt OASIS 5150 0

mpc1_opt {
    COPY mpc1_opt
}
DRC CHECK MAP mpc1_opt OASIS 5150 1
mpc2_opt {
    COPY mpc2_opt
}
DRC CHECK MAP mpc2_opt OASIS 5150 2
```

setlayer mdp_maskopt

SVRF mask data preparation command that modifies the input geometry to improve fracture quality

Usage

```
setlayer output_layer = mdp_maskopt input_layer [-max_jog_width jog_width  
-max_jog_alignment jog_alignment -min_edge_seperation sep [-max_jog_area value]  
[-exclude_region exclude_layer]]
```

Arguments

- ***output_layer***

A required keyword specifying the name of the new output target layer. The new layer is created and the results of the operation are written to the layer. This layer must be a valid Calibre layer containing polygon data.

- ***input_layer***

Specifies the name of the input layer. The layer must be a valid Calibre layer containing polygon data.

- **-max_jog_width *jog_width***

A required keyword that specifies the maximum allowable length for an edge of a polygon to be qualified as a jog.

- **-max_jog_alignment *jog_alignment***

A required keyword that specifies the maximum allowable jog movements during alignment optimization.

- **-min_edge_seperation *sep***

A required keyword that specifies the minimum required mask geometric constraints between jogs and the nearby polygonal edges of the same orientation.

- **-max_jog_area *value***

An optional keyword that specifies the maximum allowable area for a jog movement during alignment optimization. This is in square microns and can be specified in addition to the values for **max_jog_width** and **max_jog_alignment**. The default value is “off”, which means that max_jog_area is equal to (max_jog_width * max_jog_alignment).

In use, this value is set to a value smaller than (max_jog_width * max_jog_alignment). If the result of moving a jog is larger than this area, the jog is not moved. This permits more jogs to be moved (for instance, a narrow jog with a long move for alignment can still be moved) without harming image quality.

Note

 In general, max_jog_area should not be much smaller than (max_jog_width * max_jog_alignment). Values from 1/4 to 1/10 of (max_jog_width * max_jog_alignment) are recommended. Smaller values tend to consume CPU time without improving results significantly.

- -exclude_region *exclude_layer*

An optional keyword that specifies a layer that contains the geometry not to be aligned.

Description

This operation modifies the input geometry to improve fracture output quality. Constrained jog movements are applied to form polygons which when fractured will have fewer shot counts and fewer small figures.

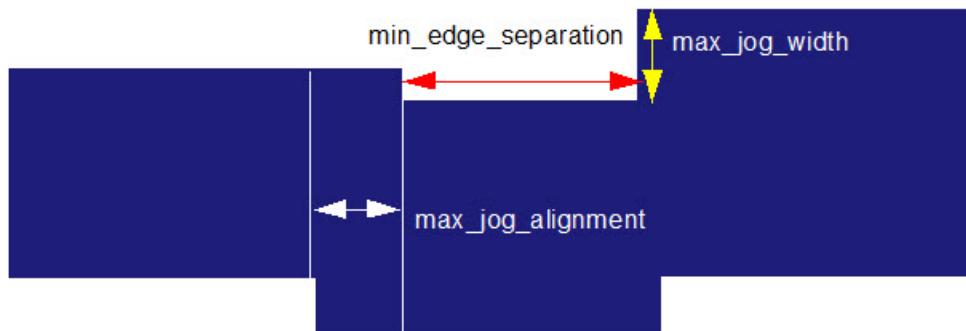
[Figure 10-1](#) illustrates where the values for max_jog_width, max_jog_alignment, and min_edge_separation are applied.

Figure 10-1. Illustration of MDP MASKOPT Arguments

max_jog_width: The maximum allowable length for an edge of a polygon to be qualified as a jog.

max_jog_alignment: The maximum allowable jog movements during alignment optimization.

min_edge_separation: The minimum required mask geometric constraints between jogs and the nearby polygonal edges of the same orientation.



Examples

```
setlayer mopt = mdp_maskopt opc_input -max_jog_width 0.010 \
    -max_jog_alignment 0.015 -min_edge_separation 0.035 \
    -max_jog_area 0.000075
```

direct_input

Enables Direct Data Entry (DDE) for Calibre MASKOPT.

Usage

```
direct_input '{  
    vboasis_path vboasis_filename [vboasis_injection [size bin_size]]  
    {[input input_number] layer number [datatype] [layer number [datatype]]...}...  
}'
```

Arguments

- **vboasis_path** *filename*

A required argument supplying the absolute path to an OASIS file name to be used as the design source. A runtime error is returned if the input file is invalid. The filename must not be in single or double quotes.

- **vboasis_injection** [size *bin_size*]

The VB:OASIS file specified using vboasis_path may contain cells that have large amounts of data and have a large extent. These large cells degrade performance in section mode processing. To overcome this problem, cells can be partitioned into bins. The size of the cells that should be binned and the size of each bin (*bin_size*) is determined automatically by default when the vboasis_injection parameter is specified.

Alternately, the *bin_size* can be explicitly specified using the size *bin_size* in microns. Any cell whose extent has a width or height more than the *bin_size* is considered for binning. The vboasis_injection creates temporary data in a directory named *<file_path>.lcb*. The binning procedure is computation-intensive and can be performed in Calibre MT or MTflex modes.

The temporary data is read or written in a directory named *<file_path>.lcb* where *file_path* is argument specified to vboasis_path. The location of this directory is same as that of *file_path*.

- {[input *input_number*] layer **number** [*datatype*]}

A required keyword set that specifies one or more layers. At least one layer declaration is required.

input input_number — An optional keyword set that specifies which input layer from the setup file to map. The index starts at 1 for the first layer listed in the setup file. The specified layer is mapped to the VB:OASIS layer **number**, allowing you to use the input layers out of order. The value you specify for *input_number* cannot exceed the number of setup layers. If this keyword set is not specified, layers are assumed to be in the same order as the setup file. Either all layers must use input *input_number*, or none of them.

layer number — A required keyword set defining a layer that is present in filename.

datatype — An optional argument that limits the datatypes that map onto the posttapeout operation's layers. If *datatype* is absent, all datatypes for that layer are used.

Description

The direct_input command enables you to use a method of direct input of OASIS files, also referred to as OASIS DDE (Direct Data Entry). OASIS DDE can accept any OASIS file. The braces {} are required.

This command works in conjunction with litho-flat mode and automatically activates it when a direct_input block is found.

This command causes the layers initially input to Calibre to be left unused. They are replaced with layers read directly from the OASIS database specified with the vboasis_path argument. Only the direct data covered by the HDB extent (as defined using a POLYGON statement in the SVRF file) is directly read from the given OASIS database. Therefore, it is not enough to pass in dummy layers through the LITHO MASKOPT statement. At least one HDB layer must contain a square large enough to cover the given OASIS file in order to ensure that everything in it is processed.

Examples

```
LITHO FILE "maskopt.setup" [
    tilemicrons 100
    processing_mode flat

    direct_input {
        vboasis_path test.oas
        input 1 layer 99
        input 2 layer 0
    }

    layer opc_input1
    layer opc_input2
    layer opc_input3
    setlayer vbo_in1 = copy opc_input1
    setlayer vbo_in2 = copy opc_input2
]

vboais_in1 { LITHO MASKOPT dummy1 dummy3 dummy4 MAP vbo_in1 \
    FILE "maskopt.setup"}
vbo_in2 { LITHO MASKOPT dummy1 dummy3 dummy4 MAP vbo_in2 \
    FILE "maskopt.setup"}
```

direct_output

Enables direct data output of OASIS files. Used with litho-flat configurations only.

Usage

```
direct_output '{'
    {vboasis_path vboasis_filename [append]
        {output name layer layer_number [layer_datatype]}{...}} [...]
        [vboasis_output_primary source]}{...}
    '}'
```

Arguments

- **vboasis_path vboasis_filename** [append]

A required argument specifying a VBOASIS filename for output. If the optional “append” argument is specified, the file is written to the end of the specified filename. If the file does not exist, it is created.

- **output name layer layer_number [layer_datatype]**

A required argument specifying one or more output layers. This argument is used for both VBOASIS and online modes. It designates an output layer that maps to a specified VBOASIS or online layer number. Adding the layer datatype is optional.

- **vboasis_output_primary source**

An optional argument that can be specified once per direct_output command. It specifies the source of the top cell name in *filename*. Use one of the following:

HDB — The topcell name from the HDB.

DIRECTIN — The top cell hname from the first **vboasis_path** input.

FILE — Same as DIRECTIN.

explicit *name* — Explicitly names the top cell.

Description

This command enables direct data output from Calibre MASKOPT. It must be run in litho-flat mode.

Enabling direct_output re-routes the specified layers mapped out of Calibre MASKOPT to a direct output file. In the output design, these layers are generated as empty layers. The layers mapped out to a direct output file must be mapped out using setlayer statements, and they cannot be optimized out by the Calibre DRC compiler (they have to be output to a dummy result file).

One or more vboasis_path blocks may be present.

All remotes must be able to access the path specified in **vboasis_filename**, and that path has to point to the same physical directory. For example, /net/machine1/export/local/out.oas should

work, but */tmp/out.oas* might not. Siemens EDA suggests using absolute paths where possible, since on some machines a path may be different from on another machine.

Limitations

- Classification blocks, properties output, and the output_window command are not supported in Calibre OPCverify scripts that use direct_output.
- A local host directory must be defined in the remote file in order for direct_output to work in Calibre MTflex mode.
- This statement can appear only once per litho setup file. Use multiple **vboasis_path** arguments in a single statement to output to multiple files.

Examples

```
direct_output {  
    vboasis_path out_data.oas  
    output input_final layer 0  
    output mopt layer 1  
}
```


Appendix A

Odd-Multiple Grid Sizing

Sizing data by an odd multiple of address units before fracturing results in that data being off-grid relative to the fracture address unit grid. This has an impact on your data and there are some possible work-arounds that allow you to retain your chosen address unit for optimum mask writing time.

Understanding Odd-Multiple Grid Sizing.....	301
Adjustments for Odd-Multiple Grid Sizing	302

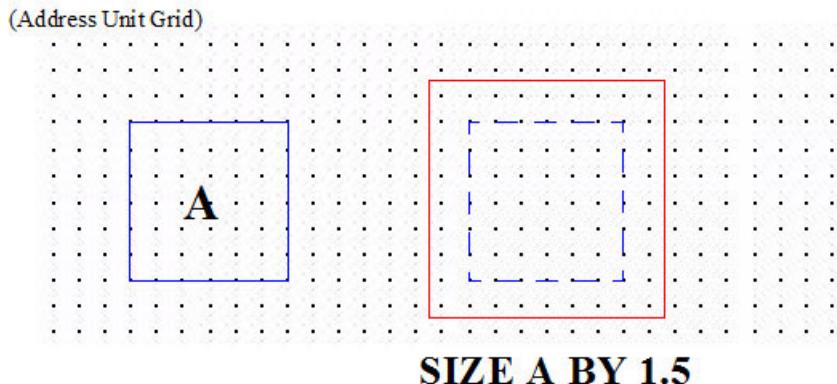
Understanding Odd-Multiple Grid Sizing

The SIZE operation sizes each edge rather than the entire polygon. Because of this, to size an entire polygon by N using the SIZE operation, you must size each edge by $(N/2)$.

This is described in Chapter 4, “[Preprocessing With SVRF](#).”

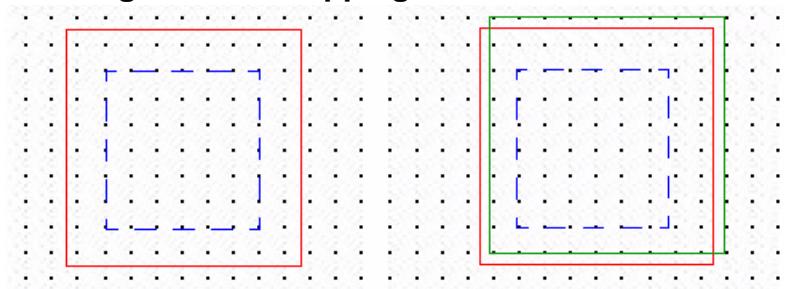
[Figure A-1](#) shows the sizing required to obtain a polygon that is 3 units larger than the original, using the operation SIZE BY (3/2). Notice the vertices are 1/2 AU off-grid relative to the address unit grid.

Figure A-1. Off-Grid Data Caused by Odd-Multiple Sizing



The Calibre FRACTURE operation automatically snaps off-grid data onto the address unit grid, shifting the data to the nearest grid point. When a vertex lies exactly between two address unit grid locations, which is the situation after odd-multiple sizing, the operation snaps it to the larger grid location, that is, up and to the right, as shown by the green shape in [Figure A-2](#).

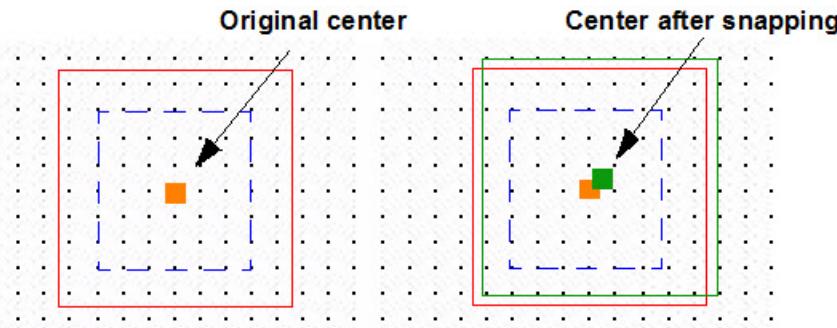
Figure A-2. Snapping Corrects Off-Grid Data



Snapping by FRACTURE

One side-effect of the snapping is that the center of the resulting figure is not in line with the center of the original data.

Figure A-3. Center-Shifting After Snapping



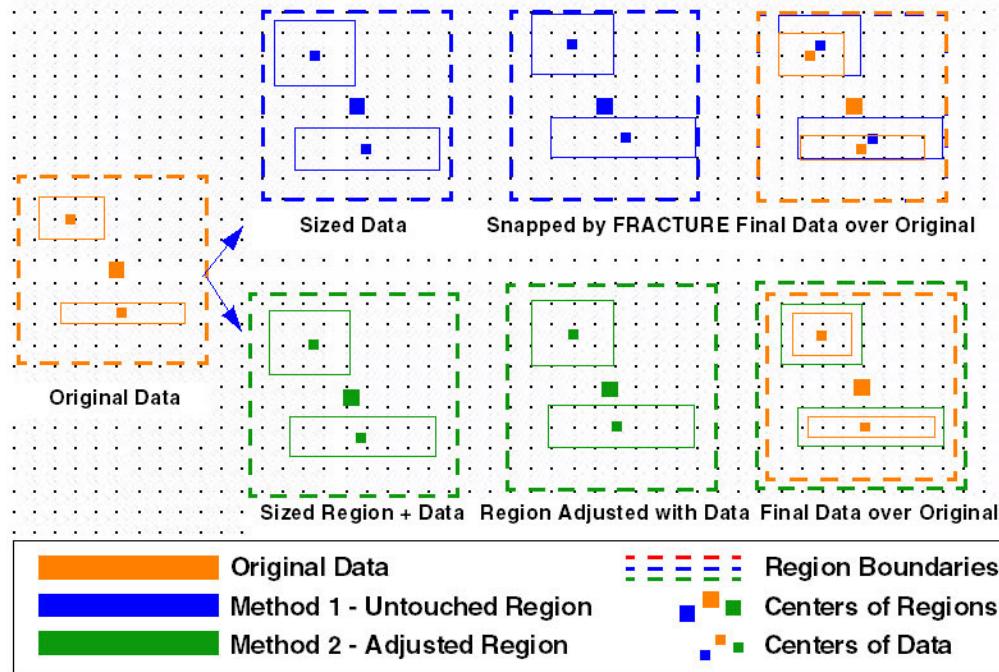
In most situations, the snapping of off-grid data after sizing does not cause any problems because the center-shifting affects all the data equally. As a result, all the centers are still in the same positions relative to one another. However, they are not in the same positions relative to the region's center, because the region itself is not snapped. As a result, when you use a job deck to combine multiple fractured files, the centers in one fractured file would be shifted relative to the centers of the other files.

Adjustments for Odd-Multiple Grid Sizing

In order to maintain alignment within the job deck, each of the fractured files must have its center in the same position relative to the centers of the fractured data as in the original data.

Because the FRACTURE operation uses the lower left corner of the region as the origin of the fracture data, you can force the fractured file to retain the original center of the data by increasing the size of the region you pass to the FRACTURE operation by 1/2 address unit in all directions. As a result of this increase, the center of the region is also offset by 1/2 an address unit. [Figure A-4](#) illustrates this behavior.

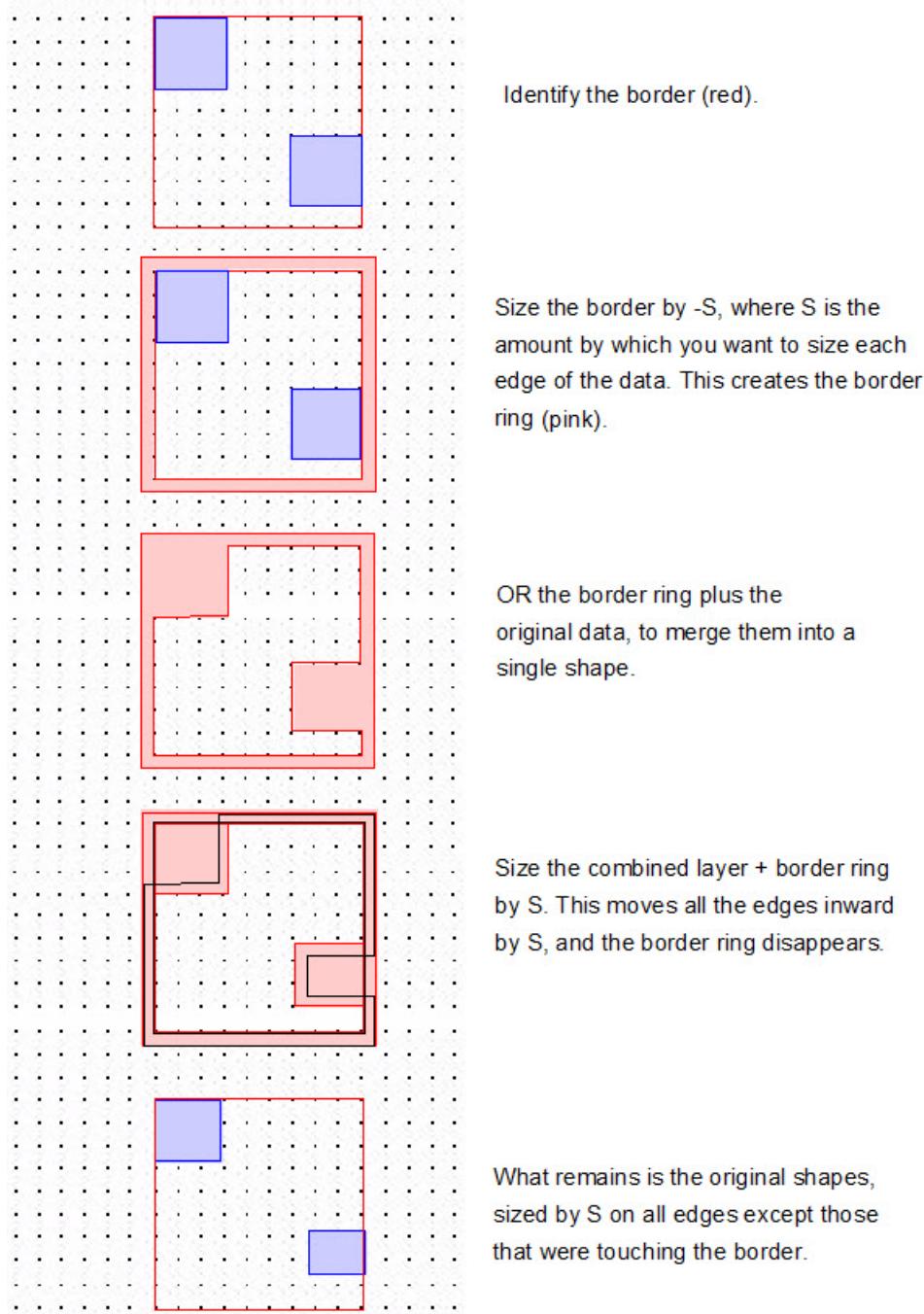
Figure A-4. Shifting the Center by Adjusting the Region



If the odd-multiple sizing you perform must also be border-aware, as described in Chapter 4, “[Preprocessing With SVRF](#),” you must make sure that you preserve the data/edge relationships at the same time that you adjust the region to correct for the center shift. This involves expanding the border by 1/2 an address unit in all directions before sizing, to adjust to the fact that region you pass to FRACTURE is larger.

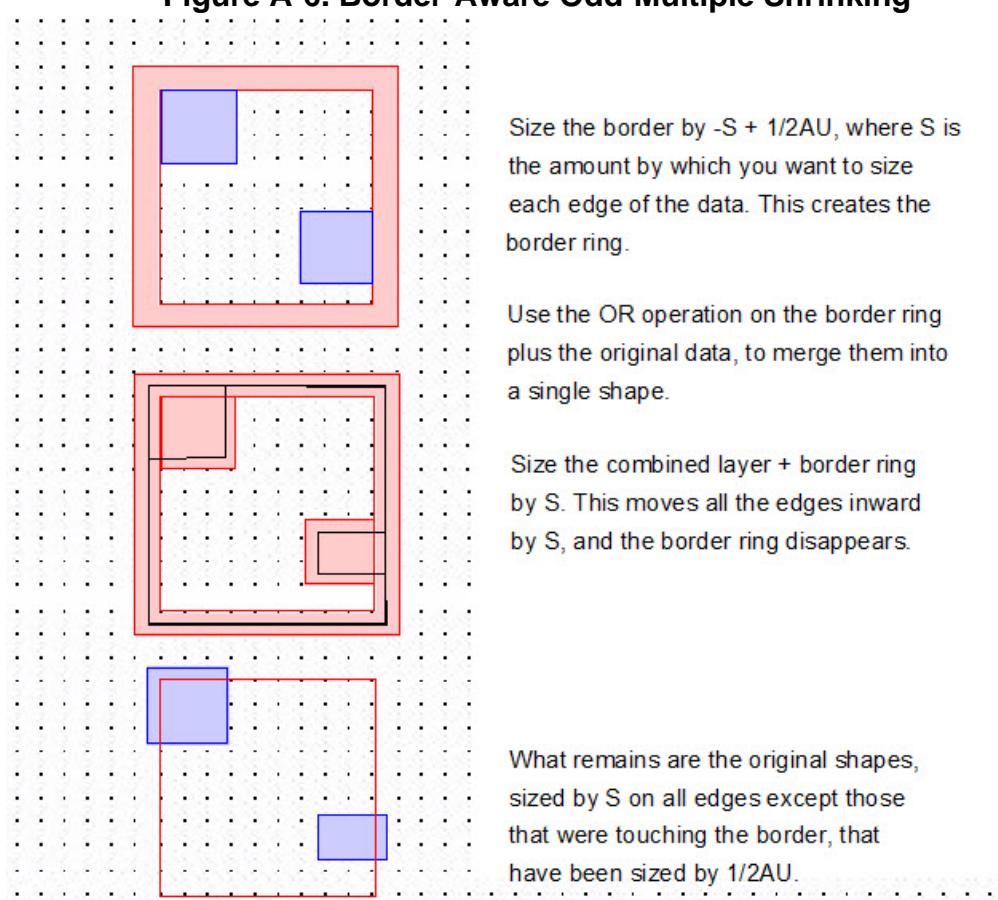
[Figure A-5](#) illustrates the process used for border-aware shrinking.

Figure A-5. Border-Aware Sizing



When correcting for the center shift caused by odd-multiple sizing and performing border-aware sizing at the same time, the border ring you create must be 1/2 an address unit wider than otherwise. This forces the geometries that touch the border to grow by 1/2 an address unit on the border-edge at the same time that they shrink along all other edges.

Figure A-6. Border-Aware Odd-Multiple Shrinking



The following SVRF excerpt performs this border-aware shrinking plus border adjustment. If you choose to use this code in your own SVRF, be sure to replace the value (S) with the actual shrink size.

```
border = EXTENT xxxx// returns bounding box of all data
border_ring = (SIZE border BY (-S + 1/2AU)) NOT border
sized_layer = SIZE (final OR border_ring) BY S
```


Appendix B

List of Exceptions

Exceptions are errors or warnings issued due to bad input or parameters. They can occur at the beginning of the program or during program execution, and can be both fatal and non-fatal. Calibre communicates exceptions by messages in the transcript and by the process exit code.

Calibre Mask Data Preparation tools issue status values indicating an error or warning has occurred during the Calibre run. These status codes result from situations where an error could not have been discovered during compilation.

MDP functions differ from the standard SVRF run time model. Normally, run time user errors are not allowed. The compiler detects all user errors before execution begins, and any failure of the program after this point is a program bug. However, certain MDP functions cannot check for all possible user errors at compile time.

For example, during an MDPVERIFY run, you could incorrectly specify an extent, but because Calibre does not access the database or fracture file during compilation this error would not be discovered until the run has begun. In this case Calibre would complete the run and print a status value at the end of the Calibre transcript. As another example, many MDP operations perform file I/O. If a file system enters a bad state during program execution, the file operation may fail, which may or may not be a fatal error.

Exceptions that occur on remotes are reported in detail in the transcript and exit code for the remote process. For MDP SVRF, they are also echoed on the primary with the prefix string “MDP REMOTE EXCEPTION:” and the name of the exception.

Transcript Messages	307
Status Values	308

Transcript Messages

Error messages always contain the string “ERROR:”, and warning messages always contain the string “WARNING:” However, messages issued from MDP commands such as FRACTURE, MDP EMBED, MDPMERGE, MDPVERIFY, and MDPSTAT use additional MDP-specific text strings.

For generic errors and warnings, these strings begin the transcript line containing the message. The Calibre MDP tools use more descriptive prefixes, formed by prepending “MDP” to the generic prefix. For instance:

- MDP runtime errors (MDP RUNTIME ERROR:)

Fatal errors cause the program to terminate abnormally. Non-fatal errors indicate that one or more SVRFs do not generate correct output.

- MDP runtime warnings (MDP RUNTIME WARNING:)

Warnings indicate a potential problem but do not necessarily indicate bad output. A generic warning message with an exit status code is also added at the end of the transcript. For example:

```
WARNING: MDP exit status: 100
```

If multiple exceptions are encountered that cause an exit, only the most severe exception is reported in the standard output. For example:

In one run, two exceptions are triggered:

- MDPVERIFY: ERROR reading file (MDP Status code 20)
- Lint error (MDP Status Code 113)

MDP outputs the most severe condition (MDP Status Code 20) in the output:

```
MDPVerify: Error reading file.
```

Refer to the transcript for more detailed information about any errors or warnings.

Status Values

The following table describes the possible status values.

Table B-1. Calibre MDP Status Values

Status Value	Description
0 (no output status)	Everything is OK. As there are no errors, no status is output.
1	Generic error.
10	FRACTURE: Runtime parse error. For example, an empty extent layer.
11	Empty database in file-to-file comparison. Since the rule deck did not require any data from HDB, the HDB was not created and MDPverify results cannot be returned to it. To remedy the situation, pass any non-empty HDB layer to MDPverify.
12	Bad OASIS DDE file. An OASIS DDE file supplied to an operation is incorrect or does not meet the requirements of the operation.
20	MDPVERIFY: Error reading file.

Table B-1. Calibre MDP Status Values (cont.)

Status Value	Description
21	MDPVERIFY: Files of the same format have different units, extents, or transformations.
22	MDPVERIFY: Comparison region and chip extents are possibly incorrect.
23	MDPVERIFY: Shift values overflowed.
24	MDPVERIFY: decompression failure.
30	In VSB11 format, there is a problem in adhering to frame file data/bde size limitations when <i>bde_filesize_limit</i> is set to 1.
31	In VSB11, the <i>frame_width</i> (height in microns) is outside the allowable range (128 to 1024 microns) when <i>frame_width_range</i> is set to 1.
32	The restrictions on a Micronic filename are violated when <i>file_name_restrictions</i> is set to 0 or +1.
41	For JEOL, one or more fields violated the HSC limit (refer to the <i>hscLimit</i> option in the FRACTURE JEOL (FRACTUREj) description for further information).
100	Generic warning.
113	Lint warning. A potential problem was detected in the input or parameters for the operation.
116	File output could not be produced in accordance with user specifications due to a conflict between parameters and input that could only be detected at run time.
130	Failed to open the index file.
131	Failed to create an index file.
132	Failed to remove an existing index file.
133	A lint check found a potential performance problem in an input OASIS file or its index.
140	OASIS bin injection data could not be read.
141	OASIS bin injection data could not be written.
142	OASIS bin injection data could not be removed.
150	Non-isothetic trapezoid data was encountered in mask data input to a verification or analysis function.

Appendix C

MDP Best Practices

This appendix contains a summary of tips, best practices, and recommendations for MDP.

General Recommendations for MDP Tools.....	312
Input Preparation for DDE	312
Indexing and Injection for OASIS Files	313
Parallel DDE Run Recommendations.....	313
Reduce Data Size to Address RAM Issues.....	314
Global Fracture Best Practices.....	315
Common Fracture Settings	315
Avoid Bottlenecks During Fracture	316
VSB Fracture and Calibre MDPview	317
Calibre FRACTUREt Best Practices.....	317
MDP EMBED Best Practices	317
DENSITY CONVOLVE Best Practices.....	318

General Recommendations for MDP Tools

There are a number of recommendations for all MDP tools.

- Using a [LAYOUT BASE LAYER](#) statement is recommended in all Calibre jobs to obtain the best performance and scalability for a variety of hierarchical layout structures. For more information, refer to the [Calibre Post-Tapeout Flow User's Manual](#).

Input Preparation for DDE.....	312
Indexing and Injection for OASIS Files	313
Parallel DDE Run Recommendations	313
Reduce Data Size to Address RAM Issues.....	314

Input Preparation for DDE

Direct Data Entry (DDE) is a method of input where you can bypass the hierarchical database constructor, HDB, and instead take input directly from a specified OASIS.MASK file using the vboasis_path command.

Several restrictions and caveats apply to the use of this alternate input method, documented in the Direct Oasis Input argument descriptions in “[Global Calibre FRACTURE Arguments](#)” on page 29.

When preparing your input for Direct Data Entry (DDE):

- The ideal data structure for input into DDE is OASIS.MASK. The input layout should use small, abutting cells of shallow hierarchy.
- If the input is not OASIS.MASK, then use bin injection instead (through the vboasis_injection command). Bin injection is for layouts containing cells that have large amounts of data and have a large extent. These large cells degrade performance in section mode processing. To overcome this issue, cells can be partitioned into bins, then loaded.

`vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]`

This keyword is described in detail in the Direct Oasis Input argument descriptions in “[Global Calibre FRACTURE Arguments](#)” on page 29.

- You should set the vboasis_injection -size option to 250.
- Use of OASIS STRICT CBLOCK format is recommended to be used in the output of jobs when the file is read by direct data input. This generally results in smaller file size and faster performance. Direct data input is invoked by the vboasis_path keyword in the [Calibre FRACTURE Syntax](#), [MDP EMBED](#), [Calibre MDPVerify](#), [DENSITY CONVOLVE](#), and [MDP OASIS_EXTENT](#) commands.

Indexing and Injection for OASIS Files

For a given OASIS input file in MDP, it is recommended that you perform pre-indexing and pre-injection on the input file.

Pre-indexing is a process where an index file is generated to speed job deck loading and viewing in Calibre MDPview. Pre-injection is a process by which large cells are partitioned into bins (bin injection) for easier loading.

These functions need only occur once for a given input file. By performing these functions outside of MDP operations in the beginning of the flow (such as Fracture, Calibre MDPverify, and MDP EMBED), you can add index and injection reusability to your automation. In addition, you can utilize your hardware more efficiently by running these functions using MT rather than relying on ad-hoc creation in a Calibre MTflex environment.

- Pre-Indexing: Use Calibre MDPview to pre-index the data. For example:

```
calibremdpv -a oasisIndex oasis_input_file -noView
```

If you intend to use the input file for MDP functions and not for viewing purposes, then use the -noView switch. This lessens the computational load by omitting some computations intended only to improve interactive viewing speed. The result is that the index is made much more quickly.

Note

 Indexing is an MT operation.

- Pre-Injection: Use Calibre MDPview to pre-inject the data. For example:

```
calibremdpv -a oasisInjection oasis_input_file
```

Note

 Injection can be run in both Calibre MT and Calibre MTflex modes.

The Calibre MDPview invocation syntax is documented in the [Calibre MDPview User's and Reference Manual](#).

Parallel DDE Run Recommendations

In UNIX, the same file cannot be written by two processes at the same time. Since indexing and bin injecting a file requires writing files whose paths are a function solely of the input file, simultaneous indexing and injection by two Calibre processes are not supported. If you want to run parallel Calibre jobs that use DDE (and by extension indexing and injection) on the same input file, care must be taken. There are two modes of operation recommended:

- Keep the input file in its original directory and put a symbolic link to the file in a different location. You can use separate run directories to invoke the separate Calibre

jobs, citing the symbolic link path as the input path to Calibre. This allows the Calibre processes to do duplicate but non-conflicting indexing and injection.

- Use the viewer oasisIndex and oasisInjection utilities to index and inject the file before any Calibre batch process uses it. The minimum of all bin sizes of any client that use the data must first be determined.

Reduce Data Size to Address RAM Issues

In cases where you experience RAM issues, such as a high remote LVHEAP or, in the worst case, remotes going into SWAP, the best method is to reduce the data being sent to the remotes. There are a number of ways you can accomplish this for Calibre Fracture, Calibre MDPverify, and MDP EMBED operations.

- In MDP EMBED, try reducing the section_size value.
- In Calibre MDPverify, try reducing the field_size value (start with 500).
- For Fracture operations:
 - For JEOL, reduce field_size (start with 500).
 - For OASIS.MASK, reduce cell_size (start with 500).
 - For Nuflare, there is no single keyword that controls section size. However:
 - For VSB12: Reduce subframe_width from the default 8 to 4. This is the most effective method in reducing remote RAM and LVHEAP.
 - Try decreasing frame_width, especially if frame splitting occurs.

Global Fracture Best Practices

There are best practices that apply to all fracture formats.

- Always use Calibre MDPverify after fracture to ensure the highest data integrity for MDP flows. Refer to Chapter 6, [Calibre MDPverify](#) for information.
- Use explicit fracture coordinates rather than the extent to avoid pattern shifting. Common methods of definition include:
 - Explicitly defining the coordinates using INSIDE OF $x1\ y1\ x2\ y2$.
 - Using the [POLYGON](#) SVRF command with coordinates, followed by INSIDE OF LAYER. This method allows for repeated reuse.
- If your rule file contains several FRACTURE operations using the same parameters, use the [FILE parameter_block_name](#) method to reuse code and ensure that all FRACTURE operations use the same parameters (refer to “[Calibre FRACTURE Syntax](#)” on page 23 for information). This is similar to the method used by the [LITHO FILE](#) command.
- In rare circumstances, fracture for any format can introduce additional grid snapping on long skew edges (non-orthogonal, non-45 degree). This affects all formats through trapezoiding (the process of converting to polygons to trapezoids). The size of the error at mask scale may be up to 1 Calibre database unit multiplied by the fracture magnification factor. The fracture output should ideally snap no more than 1/2 the fracture address unit away from the input at mask scale. For example:

UNIT LENGTH = 1e-6 meters = Calibre user unit = 1 um

PRECISION of a Calibre run = 1000 = Calibre database unit = 0.001 user unit

Fracture magnification factor = 4

Potential error at mask scale = Calibre dbu * mag factor = 0.004 um

Increasing the precision of the Calibre run can reduce the potential size of the error by the factor of the precision increase. In the example, changing the PRECISION from 1000 to 10000 reduces the potential error from 0.004 um to 0.0004 um.

Common Fracture Settings	315
Avoid Bottlenecks During Fracture	316
VSB Fracture and Calibre MDPview	317

Common Fracture Settings

There are a number of common settings used for fracture operations.

- When using Post-OPC data as input for the FRACTURE, Calibre MDPverify, and MDP EMBED operations:
 - Use DDE as this data has little to no hierarchy (flat).
 - Use section-based processing (set “computation_mode section”).
- For 45nm nodes and below, set small_value to 0.05um. This frees up the trapezoider by relaxing the constraints on it and allows the heuristics to make better decisions. It should also improve performance.
- Set shot_size to match the value of downstream mask writer. For example, for a Nuflare EBM6000 writer, use shot_size 0.8.

Avoid Bottlenecks During Fracture

To avoid bottlenecks that may occur during Fracture processing, you should use the an optimized configuration.

This includes the following recommendations:

- **Directory Structure**

Within fracture operations, there are three main directories that are created (tmp, input, and output). The tmp directory is where the temporary fracture files are kept as specified by LOCAL HOST DIR. In addition, if vboasis_injection is used, then a fourth directory (a bin injected data directory) is created with the name “*inputFileName.lcb*.” By default, the bin-injected data directory is created in the same directory specified by vboasis_path (for “on-the-fly” injection), or *inputFileName* (in the case of pre-injection using the viewer utility oasisInjection). Ideally, you should keep all four directories local.

If this is not possible, then keep both the tmp and bin-injected data directories local while storing the input and output directories over the network.

- Methods to keep the bin-injected data directory local include:
 - The location of the bin-injected data directory can be set using the MDPIIndexSearchPath environment variable (refer to the section “Setting the Location of the Index File” in the *Calibre MDPview User’s and Reference Manual* for full details).
 - Another option is to store the input file over the network, create a soft link to those files in a local working directory, and then specify the path of the local soft linked inputs when using the vboasis_path keyword.

- **Disk**

- Use fast, parallel access, RAID disks.
- Set RAID level 0 for better parallel access and no redundancy checking.

- To simulate parallel disks, use separate disks. For example:
 - For 4 disks: Disk A for tmp, disk B for the bin-injected data directory, disk C for input, disk D for output.
 - For 3 disks: Disk A for tmp and the bin-injected data directory, disk B for input, disk C for output.
 - For 2 disks: Disk A for tmp and the bin-injected data directory, disk B for input and output.

VSB Fracture and Calibre MDPview

There are a number of best practices for VSB Fracture and Calibre MDPview.

- Do not re-fracture a VSB database while viewing it in Calibre MDPview. VSB data is not completely loaded into viewer memory and viewing operations requires on-disk access. As the second fracturing operation alters the on-disk data that the viewer has already opened, it can cause the viewer to get confused about the state of the database.
- Also, do not modify a database (such as re-fracturing) in another shell while it is being viewed in Calibre MDPview. The viewer cannot appropriately reflect actions that occur outside of its own process.

Calibre FRACTUREt Best Practices

There are a number of best practices recommended for the Calibre FRACTUREt format.

- The value specified for **array_cell_cost_multiplier** value should be kept at default. However, if you run into high remote LVHEAP numbers such that the remotes are at or nearing swap, it is recommended that you decrease this value from 2 to 1. If further improvements in remote LVHEAP numbers are necessary, you can further decrease the value of this keyword.
- If scalability is an issue, it is recommended that you decrease value of **subframe_width** from the default (8) down to 4. This creates twice as many sections for FRACTURE, thus improving granularity. Changing the value of this keyword is not visible to the user in the output fracture format.

For more information, refer to “[FRACTURE NUFLARE \(FRACTUREt\)](#)” on page 66.

MDP EMBED Best Practices

Do not use **maximum_output_count** (used to limit the number of output shapes) for layer generation jobs as your results may be incomplete.

For more information, refer to “[MDP EMBED](#)” on page 126.

DENSITY CONVOLVE Best Practices

There are a number of best practices recommended for DENSITY CONVOLVE,

- When using PARTIAL $r1n\ r2n$ to specify a partial fractal kernel, it is highly recommended that you use “upsample_interpolation linear” when using partial fractal kernels to avoid “stair-stepping” when combining the partial fractal kernels.
- Although WINDOW wxy is an optional keyword, you should explicitly set the value wxy to be equal to the smallest convolution grid value cg in order to improve runtime performance.

If wxy is set to be smaller than the smallest cg , MDP spends time calculating the density values on a small grid, only to have this grid down-sampled during the convolution step. The DENSITY CONVOLVE WINDOW default (set to data extent of input) was chosen to be consistent with the DENSITY WINDOW default, but is not appropriate to use for the DENSITY CONVOLVE command.

Also, consult your process engineer for the best size of the density window. For wafer-level OPC, this value would typically be about two orders of magnitude larger than the nominal feature size. For mask-level Mask Process Correction (MPC), the rule of thumb is to set this value to 1/5 the value of the Gaussian ($wxy=1/5*s$).

For more information, refer to “[DENSITY CONVOLVE](#)” on page 250.

Index

— Symbols —

[] , 18
{ } , 18
| , 18

— A —

AND , 160

Appropriate Edges
measuring angles for , 173

Appropriate edges
definition , 171
illustration , 173
orientation of , 171

— B —

Bold words , 17

— C —

CALIBR
E_MDP_IGNORE_REDUNDANT_C
ELLS , 37
Calibre nmDRC hierarchical engine
results , 164
Calibre transcript , 164
Coincident
relative to edges , 166
relative to polygons , 166
Coincident Edge , 160
Convex Edge , 160
Copy , 159
Courier font , 18
Critical dimension
detection with MDPstat , 235

— D —

Data
edge , 166
edge cluster , 166
layer of origin , 167
merging , 166

polygon , 166
Databases
results , 164
DENSITY CONVOLVE
syntax , 250
Density operation , 249
Dimension Checks
and layer of origin , 168
data returned from , 175
ENCLOSURE , 170
example , 176
EXTERNAL , 170
identifying edge pairings , 171
illustration , 170
Internal , 170
direct_input , 297
Double pipes , 18
DRC CHECK MAP , 157
DRC MAXIMUM RESULTS , 157
DRC Results Database , 157
DRC Results Summary , 164
DRC SUMMARY REPORT , 157

— E —

Edge Clusters , 166
Edge Data , 166
Edge Pairings
measured by ENCLOSURE operation , 172
measured by External operation , 171
measured by INTERNAL operation , 171
Edges
appropriate , 171 , 173
dividing space , 166
illustration of dividing space , 166 , 167
Embedded SVRF
multiple outputs , 147
Embedding SVRF commands in a
FRACTURE statement , 119
ENCLOSURE , 160
illustration , 172

Enclosure, 170
Environment variables
 MDPIndexSearchPath, 259
Error Layers, 166
Euclidean metric
 used with dimension checks, 174
Event log, 165
Expand Edge, 161
External, 160, 170
 illustration, 171

— F —

FLAG ACUTE, 158
FLAG OFFGRID, 158
FLAG SKEW, 158
FRACTURE HITACHI, 105
FRACTURE JEOL, 49
FRACTURE MEBES, 55
FRACTURE MICRONIC, 60
FRACTURE NUFLARE, 66
FRACTURE OASIS_MBW, 110
Fracture output
 origin location, 27
FRACTUREc, 60
FRACTUREh, 44, 105
FRACTUREj, 49
FRACTUREm, 55
FRACTUREt, 67, 73, 80, 88
FullScale, 115

— G —

GDS Files
 used as Results database, 164

Grid
 calibre, 156

— H —

Heavy font, 17

— I —

INSIDE, 159
Inside
 relative to edges, 166
 relative to polygons, 166
INTERNAL, 160, 170
 illustration, 171
Italic font, 17

— L —

Layer Constructors, 160
Layer Definition Statement, 162
LAYER DIRECTORY, 156
LAYER MAP, 156
Layer of Origin, 167
Layer Selectors, 159
LAYER, SVRF statement, 156
Layout Base Layer, 155
LAYOUT ERROR ON INPUT, 146, 157
LAYOUT PATH, 154
LAYOUT PRIMARY, 155
LAYOUT SYSTEM, 155
Layout Top Layer, 155
LENGTH, 160
LITHO MASKOPT, 291

— M —

MDP DATAPREP, 282
MDP EMBED, 126
MDP Mapsize, 267
 syntax, 270, 273
MDP OASIS_EXTENT, 277
MDP_MEBES_READER_VERSION_VALU
E, 58, 59
MDPDataSearchPath, 131
MDPstat
 detecting critical dimension, 235
 overview, 235, 237
 supported formats, 235, 237
MDPverify
 aligning data automatically, 218
 comparing fractured data to fractured data,
 208
 comparing fractured data to layout
 database, 207
 comparing two VSB job decks, 215
 data transformation, 206
 MAGNIFY, 206
 MIRROR, 206
 output, 190
 result
 geometry count, 192
 XOR, 192
 REVERSE_TONE, 206

-
- ROTATE, 206
Measurement Regions
 used with dimension checks, 173
Merged Data, 166
Metrics
 default for dimension checks, 176
 used with dimension checks, 174
Minimum keyword, 18
MPCpro, 245
- N —
NUFLARE_MBF, 95
- O —
OASIS_MAPPER, 101
OASIS.MBW, 110
Opposite Extended Metric
 used with dimension checks, 175
Opposite metric
 used with dimension checks, 175
Opposite symmetric metric, 175
OR, 160
OUTSIDE, 159
Outside
 relative to edges, 166
 relative to polygons, 166
- P —
Parentheses, 18
Pipes, 18
Polygon reference data, 166
Polygons
 dividing space, 166
PRECISION, 155
pto switch, 115
- Q —
Quotation marks, 18
- R —
Reference data
 dependence on layer of origin, 168
 dimension checks and layer of origin, 170
 for edges, 166
 illustration, 168
Resizing layout elements based on location,
 267
- RESOLUTION, 155
Results database, 164
Rule files
 flag statements, 156
 layer creation statements, 161
 layer specification statements, 156
 layout statements, 154
 unit statements, 155
Runtime summary, 164
Runtime warnings, 164
- S —
Section-based processing, 117
Size operation
 MDP summary, 161
Slanted words, 17
Soft boundary, 268
Square metric
 used with dimension checks, 175
Square parentheses, 18
Status codes, 307
Status values, 307
SVRF operations
 AND, 160
 Coincident Edge, 160
 Convex Edge, 160
 Copy, 159
 ENCLOSURE, 160
 Expand Edge, 161
 EXTERNAL, 160
 INSIDE, 159
 INTERNAL, 160
 layer constructors, 160, 161
 layer selectors, 159
 LENGTH, 160
 OR, 160
 OUTSIDE, 159
 SIZE, 161
SVRF statements
 DRC CHECK MAP, 157
SVRF statements
 DRC MAXIMUM RESULTS, 157
 DRC Results Database, 157
 DRC SUMMARY REPORT, 157
 FLAG ACUTE, 158
 FLAG OFFGRID, 158

FLAG SKEW, 158
flag statements, 156
LAYER, 156
layer definition statement, 162
LAYER DIRECTORY, 156
LAYER MAP, 156
layer specification statements, 156
Layout Base Layer, 155
LAYOUT ERROR ON INPUT, 157
LAYOUT PATH, 154
LAYOUT PRIMARY, 155
layout statements, 154
LAYOUT SYSTEM, 155
Layout Top Layer, 155
PRECISION, 155
RESOLUTION, 155
UNIT LENGTH, 155
unit statements, 155

— T —

Transcript, Calibre, 164

— U —

Underlined words, 18
UNIT LENGTH, 155
Unit size
 adjusting, 27
 default settings, 27

— V —

VSB11, 67, 73, 80, 88
VSB11 scale value error, 228
VSB12, 67, 73, 80, 88

— W —

Warning messages, 165

Third-Party Information

Details on open source and third-party software that may be included with this product are available in the `<your_software_installation_location>/legal` directory.

