This document is for information and instruction purposes. Mentor Graphics 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 Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. 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 Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

U.S. GOVERNMENT LICENSE RIGHTS: The software and documentation were developed entirely at private expense and are commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Accordingly, pursuant to FAR 48 CFR 12.212 and DFARS 48 CFR 227.7202, use, duplication and disclosure by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in the license agreement provided with the software, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the owner of the Mark, as applicable. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics’ trademarks may be viewed at: www.mentor.com/trademarks.

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.

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

Send Feedback on Documentation: supportnet.mentor.com/doc_feedback_form
L-Edit 16 User Guide—Contents

1  Library and Design Configuration 43
   Database Options in L-Edit ............................................. 43
   “Design” versus “Library” ............................................. 43
   OpenAccess Database Structure ..................................... 43
      OpenAccess Cell Libraries ......................................... 44
      Cell Reservations .................................................... 44
   TDB Database Structure ............................................. 45
      TDB Cell Libraries ................................................... 45
   Launching L-Edit ....................................................... 45
      Opening an OpenAccess Database ....... 45
      lib.defs - the Library Path File in OpenAccess .................. 46
      Auto Loading TCL Scripts in an OpenAccess Database ........ 46
      File Contents in an OpenAccess Database ....................... 47
   OpenAccess Technology Libraries .................................. 48
      Protecting Your Data in Attached and Referenced Libraries .. 49
      Setting the Technology Reference in an OpenAccess Database 50
      Technology Reference Options for OpenAccess .................. 51
      OpenAccess Attached Libraries ................................. 53
   Technology Reference Options for a Tanner Database ............ 53
   Importing Technology into a Design ................................ 53
      Importing Technology from a TDB File ........................ 54
      Importing a Setup from Virtuoso .................................. 56
      Importing a Setup from Laker .................................... 56
      Importing a GDS Layer Map ........................................ 57
      GDS Layer Map File Format ........................................ 58
      Importing CDF ......................................................... 58
      Layer Merging During Imports ..................................... 59

2  Application and Design Setup 60
   Setting the Number of Available Colors ............................ 60
   Application Setup ..................................................... 61
      Application Configuration Files .................................... 61
      Workgroup and User Configuration Files ......................... 61
      Editing Configuration Files ....................................... 62
      Contents of Configuration Files ................................... 62
   Setup Application—General .......................................... 63
   Application—Editing .................................................. 65
   Setup Application—Keyboard ......................................... 67
   Setup Application—Mouse ............................................ 68
   Setup Application—Warnings ......................................... 70
   Setup Application—UPI ................................................ 71
   Setup Application—Rendering ....................................... 72
      Advanced Performance Settings ................................... 74
   Setup Application—Selection ......................................... 75
   Setup Application—Text Editor ...................................... 77
      Files Modified Outside the Text Editor .......................... 78
   Setup Application—Text Style ....................................... 79
      Adding Keywords to a Group ...................................... 80
   Design Setup .......................................................... 80
L-Edit 16 User Guide—Contents

(Listing Object Types by Layer ............................................. 123
Transferring File Information to Cells .................................. 123
Properties ............................................................................ 123
Property Types ......................................................................... 124
Binary Large Object ("Blob") Properties .................................. 124
Viewing and Editing Properties ................................................ 124
Adding a Property .................................................................... 125
Deleting a Property ................................................................... 126
Renaming a Property ................................................................ 126
Deleting a Property Value ....................................................... 126
Editing a Property Value ......................................................... 126
Adding a Copyright, Logo or Text to a File .............................. 126
Using the alphabet.tdb File ....................................................... 128
Rescaling a Design ................................................................... 130
Exiting L-Edit .......................................................................... 132
Abnormal Termination .............................................................. 132

4 Importing and Exporting Files ............................................. 133
Importing Files ................................................................. 133
Importing GDS Files ............................................................ 133
Prompt if unknown layers are found ....................................... 134
Setting the View Level ......................................................... 135
Importing CIF Files ............................................................. 136
Prompt if unknown layers are found ....................................... 138
Importing OASIS Files .......................................................... 137
Importing DXF Files ............................................................. 139
Draw > Convert > Connect Segments ....................................... 140
Importing Gerber Files .......................................................... 142
Importing Graphics .............................................................. 142
Merging and Dithering Options ............................................. 143
Exporting Files ....................................................................... 144
Exporting GDS Files ............................................................ 144
Mapping the GDS Data Type and Purpose ............................. 145
Polygons with Too Many Vertices ......................................... 145
Exporting CIF Files ............................................................. 148
Exporting OASIS Files .......................................................... 148
Exporting DXF Files ............................................................. 150
Exporting Gerber Files .......................................................... 152
Exporting PostScript Masks .................................................. 153
CIF File Formatting ............................................................. 153
Symbols ................................................................................ 153
Calls (Instances) .................................................................... 154
Geometric Primitives ............................................................ 154
Layers ..................................................................................... 156
Fabrication Cell ...................................................................... 156
Restrictions ............................................................................ 157
Extensions ............................................................................... 157
Wires in CIF ............................................................................ 157
Scaling ..................................................................................... 158
GDSII File Formatting ........................................................... 158
GDSII Properties ..................................................................... 158
GDSII Naming ......................................................................... 160
L-Edit 16 User Guide—Contents

(Continued)

GDSII Date Formats .............................................................. 161
GDSII Shape Definition ...................................................... 161
GDSII Data Type ................................................................. 161
Assigning Data Types ......................................................... 162
Wires in GDSII ................................................................. 162
Setting GDSII Port and Port Text Size .................................. 163

5 The L-Edit Interface ......................................................... 164

Elements of the User Interface .............................................. 164
Menus ................................................................................. 164
The Window Menu ........................................................... 165
The Help Menu ................................................................. 165
Opening the Documentation ............................................... 165
Installing Examples .......................................................... 166
Managing Commuter Licensing ......................................... 166
Diagnostic Tools for Customer Support .............................. 168
Displaying the Product Version or Contacting Customer Support ........................................ 169
Design Navigator ............................................................ 169
Toolbars .............................................................................. 170
Dockable Toolbars and Views ............................................. 170
Pinning A Docked View ...................................................... 172
Standard Toolbar .............................................................. 173
Drawing Toolbar ............................................................... 173
Editing Toolbar ................................................................. 174
Verification Toolbar .......................................................... 175
Compact Layer Palette ...................................................... 175
Base Point Toolbar ........................................................... 175
Object Snap Toolbar .......................................................... 176
Node Highlighting Toolbar .................................................. 176
Database Toolbar ............................................................ 177
Alignment Toolbar ............................................................ 177
MultiGrid Toolbar ............................................................. 177
Customizing Toolbars ....................................................... 178

This opens the Customize dialog, to the Toolbars tab. Note that in this dialog the checkmarks control only whether or not a toolbar is displayed. The buttons apply only to the toolbar that is highlighted, and will be applied even if a toolbar is not currently displayed. ............................................. 178

Adding a Command to a Toolbar ......................................... 178
Use the Commands tab to add a button for an existing command to any toolbar ............................................. 179

Adding a New Toolbar ....................................................... 179
Adding a New Command Button ......................................... 180
Example: Adding a Button to Toggle Snap Grid Size .............. 180
Adding a Button using the tcl workspace command ............. 182

Docking Views ................................................................. 182
Libraries Navigator ........................................................... 182
Layer Palette ................................................................. 183
and Log Window ............................................................. 183
Verification Navigator ....................................................... 183
SDL Navigator ................................................................. 183
Aerial View ................................................................. 183

Status Bars ......................................................................... 184
Status Bar ................................................................. 184
Mouse Button Bar ........................................................... 185
# L-Edit 16 User Guide—Contents

## 6 Viewing the Layout

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controlling Layer Display</td>
<td>181</td>
</tr>
<tr>
<td>Sorting Layers</td>
<td>181</td>
</tr>
<tr>
<td>Hiding Layers</td>
<td>182</td>
</tr>
<tr>
<td>Show/Hide Layers Using the View &gt; Layers Command</td>
<td>182</td>
</tr>
<tr>
<td>Show/Hide Layers Using the Mouse</td>
<td>182</td>
</tr>
<tr>
<td>Locking Layers</td>
<td>183</td>
</tr>
<tr>
<td>Protecting Layers</td>
<td>184</td>
</tr>
<tr>
<td>Using the Compact Layer Palette</td>
<td>184</td>
</tr>
<tr>
<td>Using the Layer Palette</td>
<td>186</td>
</tr>
<tr>
<td>Buttons in the Layer Palette Header</td>
<td>186</td>
</tr>
<tr>
<td>Right-Click Menu in the Layer Palette</td>
<td>186</td>
</tr>
<tr>
<td>Creating and Saving a Layer Palette</td>
<td>186</td>
</tr>
<tr>
<td>Viewing Layout Hierarchy</td>
<td>186</td>
</tr>
<tr>
<td>Displaying Instance Insides</td>
<td>186</td>
</tr>
<tr>
<td>Displaying Instance Insides While Drawing and Editing</td>
<td>186</td>
</tr>
<tr>
<td>Zooming and Panning</td>
<td>186</td>
</tr>
<tr>
<td>Zooming</td>
<td>186</td>
</tr>
<tr>
<td>Zooming While Editing In-Place</td>
<td>186</td>
</tr>
<tr>
<td>Panning</td>
<td>186</td>
</tr>
<tr>
<td>Pan To Cell Edge</td>
<td>186</td>
</tr>
<tr>
<td>Object Pan Settings</td>
<td>186</td>
</tr>
<tr>
<td>Zooming and Panning with the Mouse</td>
<td>186</td>
</tr>
<tr>
<td>Mouse Wheel Functions</td>
<td>186</td>
</tr>
<tr>
<td>Auto-Panning</td>
<td>186</td>
</tr>
<tr>
<td>Moving to Specific Coordinates</td>
<td>186</td>
</tr>
<tr>
<td>Toggling Views</td>
<td>186</td>
</tr>
<tr>
<td>Using the Multigrid Toolbar to Toggle Grid Size</td>
<td>186</td>
</tr>
<tr>
<td>Schematic Probing and Selection Synchronization</td>
<td>186</td>
</tr>
<tr>
<td>Cross-Section Viewer</td>
<td>186</td>
</tr>
<tr>
<td>Grow/Deposit</td>
<td>186</td>
</tr>
<tr>
<td>Etch</td>
<td>186</td>
</tr>
<tr>
<td>Implant/Diffuse</td>
<td>186</td>
</tr>
<tr>
<td>Operation</td>
<td>186</td>
</tr>
<tr>
<td>Display</td>
<td>186</td>
</tr>
<tr>
<td>Single-Step Display</td>
<td>186</td>
</tr>
<tr>
<td>Process Definition Files</td>
<td>186</td>
</tr>
<tr>
<td>Example</td>
<td>186</td>
</tr>
</tbody>
</table>

## 7 Cell Hierarchy and Views

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Views</td>
<td>186</td>
</tr>
<tr>
<td>Cell Information</td>
<td>186</td>
</tr>
<tr>
<td>The Libraries Navigator</td>
<td>186</td>
</tr>
<tr>
<td>The Cell List Filter Menu</td>
<td>218</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----</td>
</tr>
<tr>
<td>Cell List Filter Settings</td>
<td>218</td>
</tr>
<tr>
<td>How to Set Column Headings in the Cell List</td>
<td>220</td>
</tr>
<tr>
<td>Saving Column Heading Setups 1, 2 and 3</td>
<td>221</td>
</tr>
<tr>
<td>Setting Colors in the Cell List</td>
<td>222</td>
</tr>
<tr>
<td>Highlighted vs Selected in the Cell List</td>
<td>222</td>
</tr>
<tr>
<td>Right-Click Menu in the Cell List</td>
<td>223</td>
</tr>
<tr>
<td>Reserving a Cell</td>
<td>224</td>
</tr>
<tr>
<td>Explicit Reservation</td>
<td>225</td>
</tr>
<tr>
<td>The Database Toolbar</td>
<td>226</td>
</tr>
<tr>
<td>Implicit Reservation</td>
<td>226</td>
</tr>
<tr>
<td>Unreserving a Cell</td>
<td>227</td>
</tr>
<tr>
<td>Listing of Reserved Cells</td>
<td>227</td>
</tr>
<tr>
<td>Creating a New Cell</td>
<td>227</td>
</tr>
<tr>
<td>Opening a Cell</td>
<td>229</td>
</tr>
<tr>
<td>Updating a Cell</td>
<td>229</td>
</tr>
<tr>
<td>Reverting a Cell</td>
<td>230</td>
</tr>
<tr>
<td>Saving a Cell</td>
<td>231</td>
</tr>
<tr>
<td>Renaming or Moving a Cell</td>
<td>231</td>
</tr>
<tr>
<td>Closing a Cell with a New Name</td>
<td>232</td>
</tr>
<tr>
<td>Saving a Single Cell to Another File</td>
<td>233</td>
</tr>
<tr>
<td>Copying a Piece of a Cell to Another Cell</td>
<td>233</td>
</tr>
<tr>
<td>Database Evaluation When Copying or Deleting Cells</td>
<td>234</td>
</tr>
<tr>
<td>Copying a Cell</td>
<td>236</td>
</tr>
<tr>
<td>Copying to a different database</td>
<td>238</td>
</tr>
<tr>
<td>Deleting a Cell</td>
<td>239</td>
</tr>
<tr>
<td>Listing the Object Types or Layers Used in a Cell</td>
<td>241</td>
</tr>
<tr>
<td>Cell &gt; Cell Object Summary</td>
<td>241</td>
</tr>
<tr>
<td>Cell &gt; Layer-Cell Cross Reference</td>
<td>242</td>
</tr>
<tr>
<td>The Design Navigator</td>
<td>242</td>
</tr>
<tr>
<td>Design Navigator Display Modes</td>
<td>244</td>
</tr>
<tr>
<td>Top down - all cells</td>
<td>245</td>
</tr>
<tr>
<td>Bottom up - all cells</td>
<td>245</td>
</tr>
<tr>
<td>Top down - non-instanced</td>
<td>246</td>
</tr>
<tr>
<td>By date modified</td>
<td>246</td>
</tr>
<tr>
<td>DRC Status</td>
<td>247</td>
</tr>
<tr>
<td>DRC Status Icons</td>
<td>247</td>
</tr>
<tr>
<td>Icons in the Design Navigator</td>
<td>247</td>
</tr>
<tr>
<td>Performing Cell Operations with the Design Navigator</td>
<td>248</td>
</tr>
<tr>
<td>Copying and Instancing from the Design Navigator</td>
<td>249</td>
</tr>
<tr>
<td>Instancing Cells</td>
<td>249</td>
</tr>
<tr>
<td>Toggling Instance Selectability</td>
<td>249</td>
</tr>
<tr>
<td>Creating Instances</td>
<td>251</td>
</tr>
<tr>
<td>Editing Instances</td>
<td>252</td>
</tr>
<tr>
<td>Editing an Original Instance In-Place</td>
<td>253</td>
</tr>
<tr>
<td>&quot;Push to Object&quot;</td>
<td>253</td>
</tr>
<tr>
<td>Edit Object(s) for Instances</td>
<td>254</td>
</tr>
<tr>
<td>Creating Instance Arrays</td>
<td>256</td>
</tr>
<tr>
<td>Creating Arrays using the Mouse</td>
<td>256</td>
</tr>
<tr>
<td>Flattening an Instance</td>
<td>257</td>
</tr>
<tr>
<td>Assigning Names to Multiple Instances</td>
<td>258</td>
</tr>
<tr>
<td>Setting the Scope of Action for Multiple Cell Operations</td>
<td>260</td>
</tr>
<tr>
<td>Replacing Instances</td>
<td>261</td>
</tr>
</tbody>
</table>
You can replace instances of cells from one library with instances of like named cells from a different library using the **Replace Instances > Retarget Library** command. First select the cells in the library navigator that you wish to replace, then invoke the Retarget Library command and specify the scope in which you want to replace instances, and the new library where the new cells are located. Cell-views in the specified scope will be replaced with cells with the same cell and view name from the new library.

### 8 Drawing Objects

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object Types</strong></td>
<td>278</td>
</tr>
<tr>
<td><strong>Selecting a Layer</strong></td>
<td>279</td>
</tr>
<tr>
<td><strong>Selecting a Drawing Tool</strong></td>
<td>279</td>
</tr>
<tr>
<td>Selecting Angle Constraints for Drawing Tools</td>
<td>280</td>
</tr>
<tr>
<td>Changing Angle Constraints while Drawing or Editing</td>
<td>280</td>
</tr>
<tr>
<td>Drawing in Outline Mode</td>
<td></td>
</tr>
<tr>
<td><strong>Drawing Objects</strong></td>
<td>280</td>
</tr>
<tr>
<td>Boxes</td>
<td>281</td>
</tr>
<tr>
<td>Circles</td>
<td>281</td>
</tr>
<tr>
<td>Pie Wedges</td>
<td>281</td>
</tr>
<tr>
<td>Tori</td>
<td>281</td>
</tr>
<tr>
<td>Polygons and Wires</td>
<td>281</td>
</tr>
<tr>
<td>Self-Intersecting Polygons and Wires</td>
<td>282</td>
</tr>
<tr>
<td>Self-Intersecting Polygons</td>
<td>283</td>
</tr>
<tr>
<td>Ambiguous Fill Polygons</td>
<td>283</td>
</tr>
<tr>
<td>Winding Number</td>
<td>284</td>
</tr>
<tr>
<td>Curves</td>
<td>284</td>
</tr>
<tr>
<td>Converting a Straight Polygon Edge to a Curve</td>
<td>285</td>
</tr>
<tr>
<td>Curve Height</td>
<td>286</td>
</tr>
<tr>
<td>Chamfers and Fillets</td>
<td>286</td>
</tr>
<tr>
<td>Setting the Chamfer/Fillet Distance or Radius</td>
<td>287</td>
</tr>
<tr>
<td><strong>Electrical Ports</strong></td>
<td>289</td>
</tr>
<tr>
<td>Drawing and Editing Ports</td>
<td>289</td>
</tr>
<tr>
<td>Adding Additional Port Geometry</td>
<td>291</td>
</tr>
<tr>
<td>Terminals and Pins</td>
<td>291</td>
</tr>
<tr>
<td>Legacy Ports</td>
<td>291</td>
</tr>
<tr>
<td>Legacy Layers</td>
<td>292</td>
</tr>
<tr>
<td><strong>Text Labels</strong></td>
<td>292</td>
</tr>
<tr>
<td><strong>Rulers</strong></td>
<td>293</td>
</tr>
<tr>
<td>Editing and Deleting Rulers</td>
<td>294</td>
</tr>
<tr>
<td>Edit Object(s) for Rulers</td>
<td>294</td>
</tr>
<tr>
<td><strong>Wire Styles</strong></td>
<td>296</td>
</tr>
<tr>
<td>End Styles and Join Styles</td>
<td>297</td>
</tr>
<tr>
<td>Wire Style Defaults</td>
<td>298</td>
</tr>
</tbody>
</table>
Wire Utilities for Editing Wires .................................................. 299
Adding a Wire Section ................................................................. 299
Joining Wires ................................................................. 299
Extending Wires ................................................................. 301
Resume Drawing a Wire from its Endpoint. .................................. 302
Slicing Wires ................................................................. 303
Fixing Wires With Short Segments ............................................. 303
Object Snapping ........................................................................ 304
Snapping Existing Geometry to the Manufacturing Grid ............... 305
Vertex Snapping ....................................................................... 306
Intersection Snapping ................................................................. 306
Object Snap Setup ....................................................................... 307
Object Snap Toolbar ................................................................. 309
  Object Snap Shortcut Keys .................................................... 310
  Order of Preference in Snapping .............................................. 310
Selecting Objects ...................................................................... 311
  Explicit Selection .................................................................. 311
  Using a Selection Box .............................................................. 311
  Disallowing Instance Selection .............................................. 311
  Implicit Selection .................................................................. 312
  Extend Selection .................................................................... 312
  Cycle Selection ....................................................................... 312
  Edge Selection ....................................................................... 312
  Universal Selection .................................................................. 313
Deselecting Objects .................................................................... 313
  Explicit Deselection ............................................................... 313
  Implicit Deselection ............................................................... 314
  Hidden Deselection ............................................................... 314
  Universal Deselection ............................................................. 314
Deleting Objects ........................................................................ 314
Undoing or Redoing an Operation ................................................ 314
  Undo ................................................................................... 314
  Redo ................................................................................... 315
Grouping and Ungrouping Objects ................................................ 315
  Grouping Instances to Create an Array .................................... 316
  Ungrouping Instances ............................................................. 317
  Undoing Draw > Group and Draw > Ungroup ......................... 317

9 Editing Objects ..................................................................... 318
Finding Objects ......................................................................... 318
  Find Next/Find Previous ....................................................... 319
Hierarchical Find ....................................................................... 319
Moving Objects .......................................................................... 320
  Repositioning ....................................................................... 321
    Move By Options ............................................................... 322
    Nudge ............................................................................... 324
    Force Move Mode .............................................................. 324
    Base Point Mode ............................................................... 324
      Setting the Base Point ...................................................... 325
      Move and Copy/Paste Operations in Base Point Mode .......... 325
  Reorienting ........................................................................... 326
    Specifying Rotation Parameters ......................................... 326
Moving Objects from One Layer to Another ........................................... 327

Copying and Duplicating Objects ......................................................... 328
  Copying Objects ........................................................................... 328
  Duplicating Objects .................................................................... 328
  Copying Images .......................................................................... 328

Pasting Objects .................................................................................. 328
  Paste to Cursor Feature ................................................................. 329

Resizing and Reshaping ..................................................................... 329
  Types of Edge Moves ................................................................... 329
    Standard Move - Preserve Edge Length, Change Angle ................ 330
    Orthogonal Edge Move ............................................................... 330
    Two Options for a Perpendicular Edge Move ............................ 330
    Perpendicular Edge Move - Change Length, Preserve Angle .... 331
  Editing Multiple Objects Simultaneously ...................................... 331
    Adding Vertices ......................................................................... 332
    Pie Wedges and Tori ................................................................. 332
    Resizing Text ............................................................................ 332

Measuring Distance Between Objects ............................................... 333
  Temporary Rulers ......................................................................... 333
  Tools > Measure .......................................................................... 333
    Measuring Between Objects at the Same Level of Hierarchy ...... 334
    Measuring Between Objects on Different Levels of Hierarchy ... 334

Aligning and Distributing Objects ...................................................... 335
  Global Alignment and Distribution Options ................................. 336
    Minimum and Abutment Bounding Boxes ................................. 337
  Alignment Commands ................................................................. 337
  Global Distribution Options .......................................................... 339
  Distribution Commands ............................................................... 341
  Tiling Options ............................................................................. 341
  Tile Commands ............................................................................ 342

Slicing ............................................................................................... 343

Nibbling ........................................................................................... 344

Merging Objects ............................................................................... 345

Boolean and Grow Operations .......................................................... 345

Converting Curved and other Objects to Polygons ......................... 347

Converting All-Angle Polygons to Curved Polygons ....................... 348

Converting Polygons to Orthogonal or 45° Geometry ...................... 348

Removing Curves from Polygons ...................................................... 350

Fracturing Polygons ......................................................................... 350

Editing Objects From a Dialog Window ............................................ 351

Edit > Edit Object(s) ........................................................................ 351
  Common Text Edit Fields .............................................................. 351
  Editing Multiple Object Using Text .............................................. 351
    Mixed-Value and Disabled Fields ............................................. 352
    Stepping Through Selected ObjectsOne at a Time ..................... 352

Edit Object(s) for Boxes ................................................................... 354

Edit Object(s) for Polygons ............................................................... 355

Edit Object(s) for Wires ................................................................... 356

Edit Object(s) for Circles ................................................................. 357

Edit Object(s) for Pie Wedges .......................................................... 358

Edit Object(s) for Tori ...................................................................... 359

Edit Object(s) for Ports ..................................................................... 360

Edit Object(s) for Text Labels .......................................................... 361
## Editing

- Opening the Command Window .................. 362
- Using the Command Window ................. 362

### TCL Commands ................... 363

#### Syntax ...................... 364

- Coordinate Entry Options ............... 364
- Command Completion Using a Mouse Click .... 365
- Reference Point Location ............. 365
  - Reference Point Location for Multiple Objects ... 365
- Special Characters ................... 366
- Keyboard Shortcuts ................. 366

### Command Reference ............... 366

- !! ....................................... 367
- ↑, ↓ ..................................... 367
- <Esc> .................................. 367
- Array .................................... 367
- Box ..................................... 368
- Copy ..................................... 369
- Goto ..................................... 369
- Instance ................................. 369
- Layer .................................... 370
- Move ..................................... 370
- Path ..................................... 370
- Paste ..................................... 371
- Polygon ................................... 371
- Rotate ................................... 372
- Run ...................................... 372
- Text ..................................... 372
- Width ..................................... 373

### Command Scripting ............... 373

## 10 Parameterized Cells  ............... 374

### Creating T-Cells ................. 374

- H-Stretch and VStretch Parameter Types for Edge Selection .... 376
- T-Cell Code Templates ................... 377

### TCL Commands Specific to L-Edit .... 379

- Opening T-Cells ................... 380
- Closing T-Cells ..................... 381
- Instancing T-Cells ................. 381
- Regenerating T-Cell Instances ......... 382
- T-Cell Callbacks ................. 383

### Generating T-Cell Code from Layout —“T-Cell Builder” ............... 384

- Defining Stretch Ports for the T-Cell Builder  ............... 385
  - Constructing a T-Cell with Stretch Parameters—MOSFET Example ....... 385
  - Parameter Types in the T-Cell Builder .................. 387
  - Repeating Elements with the T-Cell Builder .................. 388
  - Setting the Layer as a T-Cell Builder Parameter .............. 388
  - Defining Conditional Inclusion as a T-Cell Builder Parameter ...... 390
  - Finding Objects that have T-Cell Builder Parameters ............. 391

### Generating Via Cells ............... 391

- Standard Vias ................... 391
- Custom Vias ..................... 392
# L-Edit 16 User Guide—Contents

(Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Via Icons</td>
<td>392</td>
</tr>
<tr>
<td>Defining a Standard Via</td>
<td>393</td>
</tr>
<tr>
<td>Defining a Custom Via</td>
<td>394</td>
</tr>
<tr>
<td>Editing a Via</td>
<td>395</td>
</tr>
<tr>
<td>Placing a Generated Via</td>
<td>395</td>
</tr>
<tr>
<td><strong>Valid Vias</strong></td>
<td>396</td>
</tr>
<tr>
<td>Using Valid Vias for Assisted Routing</td>
<td>398</td>
</tr>
<tr>
<td>Routing Example</td>
<td>398</td>
</tr>
<tr>
<td>How to Use Assisted Routing</td>
<td>399</td>
</tr>
<tr>
<td>Editing a Via During Placement</td>
<td>399</td>
</tr>
<tr>
<td>Cycling Through Multiple Vias with the Same Layer Definitions</td>
<td>399</td>
</tr>
<tr>
<td>'Undo' During Assisted Routing</td>
<td>400</td>
</tr>
<tr>
<td>Placing an Array with Assisted Routing</td>
<td>400</td>
</tr>
<tr>
<td><strong>Generating Guard Rings</strong></td>
<td>400</td>
</tr>
<tr>
<td>Defining a Guard Ring</td>
<td>401</td>
</tr>
<tr>
<td>Using the Fracture Option</td>
<td>402</td>
</tr>
<tr>
<td>Using the Area Option</td>
<td>403</td>
</tr>
<tr>
<td>Editing a Guard Ring Cell</td>
<td>403</td>
</tr>
<tr>
<td>Placing a Guard Ring</td>
<td>403</td>
</tr>
<tr>
<td>Deleting a Guard Ring</td>
<td>405</td>
</tr>
<tr>
<td><strong>11 Automatic Device Generation</strong></td>
<td>406</td>
</tr>
<tr>
<td>The DevGen Process</td>
<td>406</td>
</tr>
<tr>
<td>Establish the Process Technology</td>
<td>406</td>
</tr>
<tr>
<td>Define the Device</td>
<td>406</td>
</tr>
<tr>
<td>Generate Device Layout</td>
<td>407</td>
</tr>
<tr>
<td>Edit a Device Class or Create a New One</td>
<td>407</td>
</tr>
<tr>
<td>The HiPer DevGen Process</td>
<td>407</td>
</tr>
<tr>
<td>Establish the Process Technology</td>
<td>407</td>
</tr>
<tr>
<td>Define the Device</td>
<td>407</td>
</tr>
<tr>
<td>Generate Device Layout</td>
<td>407</td>
</tr>
<tr>
<td>Generate Device Layout from an SDL Netlist</td>
<td>407</td>
</tr>
<tr>
<td><strong>Standard DevGen Devices</strong></td>
<td>408</td>
</tr>
<tr>
<td>DevGen Capacitor</td>
<td>408</td>
</tr>
<tr>
<td>DevGen Diode</td>
<td>408</td>
</tr>
<tr>
<td>DevGen Inductor</td>
<td>408</td>
</tr>
<tr>
<td>DevGen MOSFET</td>
<td>408</td>
</tr>
<tr>
<td>DevGen Resistor</td>
<td>408</td>
</tr>
<tr>
<td>Editing a Device Class</td>
<td>408</td>
</tr>
<tr>
<td><strong>HiPer DevGen Devices</strong></td>
<td>409</td>
</tr>
<tr>
<td>HiPer DevGen Current Mirror</td>
<td>409</td>
</tr>
<tr>
<td>Current Mirror Definition - Page 1</td>
<td>410</td>
</tr>
<tr>
<td>Current Mirror Definition - Page 2</td>
<td>412</td>
</tr>
<tr>
<td>Generate a Current Mirror</td>
<td>415</td>
</tr>
<tr>
<td>HiPer DevGen Differential Pair</td>
<td>415</td>
</tr>
<tr>
<td>Differential Pair Definition - Page 1</td>
<td>416</td>
</tr>
<tr>
<td>Differential Pair Definition - Page 2</td>
<td>418</td>
</tr>
<tr>
<td>Generate a Differential Pair</td>
<td>419</td>
</tr>
<tr>
<td>Multiple Combinations of Matching Requirements</td>
<td>420</td>
</tr>
<tr>
<td>HiPer DevGen MOSFET</td>
<td>421</td>
</tr>
<tr>
<td>MOSFET Definition</td>
<td>421</td>
</tr>
<tr>
<td>Generate a MOSFET</td>
<td>423</td>
</tr>
<tr>
<td>HiPer DevGen MOSFET Array</td>
<td>423</td>
</tr>
<tr>
<td>MOSFET Array Definition</td>
<td>424</td>
</tr>
<tr>
<td>Generate a HiPer MOSFET Array</td>
<td>426</td>
</tr>
<tr>
<td>HiPer DevGen Resistor</td>
<td>428</td>
</tr>
<tr>
<td>Resistor Page 1</td>
<td>428</td>
</tr>
</tbody>
</table>
L-Edit 16 User Guide—Contents

Device Generation From an SDL Netlist ........................................ 432
  Schematic Characteristics that Define a HiPer DevGen Device .... 439
  Schematic Criteria for a Current Mirror ................................. 440
  Schematic Criteria for a Differential Pair ............................. 440
  Schematic Criteria for a Resistor Array ................................. 440

HiPer DevGen Process Setup .................................................... 441
  Layer Assignments ........................................................... 442
  Design Rule Assignments ................................................... 443

12 Derived Layers ................................................................. 444
  Layer Derivation Steps ....................................................... 444
  Defining Derived Layers ..................................................... 445
    Boolean Layer Derivations ............................................... 446
      Boolean Operation Outcomes ......................................... 447
      AND ................................................................. 447
      OR ................................................................. 447
      NOT ............................................................. 447
      Grow ............................................................... 448
      Boolean Order of Operations ....................................... 449
    Select Layer Derivations ................................................. 450
      Select Operation Relationships .................................... 451
        Inside ......................................................... 451
        Not Inside ................................................... 452
        Outside ....................................................... 452
        Not Inside ................................................... 453
        Hole ........................................................... 453
        Cut ............................................................. 454
        Not Cut ....................................................... 454
        Touch .......................................................... 455
        Not Touch .................................................... 455
        Enclose ......................................................... 455
        Overlap ......................................................... 455
        Vertex ............................................................ 456
      Area Layer Derivations ............................................... 456
      Density Layer Derivations ......................................... 457
    Setting Up Command File Derived Layers .............................. 458
  Generating Derived Layers .................................................. 458
    Choose a Command Set ................................................... 458
    Run the Derivation ...................................................... 458
    Automatic Layer Generation with DRC and Extract .................. 460
    Showing and Hiding Generated Layers ................................ 461
    Locking Generated Layers .............................................. 461
    Removing Generated Layers ............................................. 461

13 Schematic-Driven Layout (SDL) ............................................. 462
  Assisted Place and Route .................................................. 462
    Using the Assisted Router .............................................. 463
      Configure for Optimal Routing ..................................... 463
      Import a Netlist ................................................... 464
      Position Layout .................................................... 464
      Route Critical Nets, Auto-Route Remaining Nets ................. 464
      Check for Shorted Nets ............................................. 464
      Enter Engineering Change Orders (ECOs) .......................... 464
    The SDL Navigator ....................................................... 464
The SDL Navigator Toolbar ........................................... 465
Right-Click Menu Commands ..................................... 467
The SLD Navigator Command Menu ............................ 469
Tools > SDL Navigator Commands .............................. 470
Importing a Netlist ..................................................... 471
   Updating from an Engineering Change Order (ECO) ........ 473
Automatically Generating Layout Elements ................. 474
Adding Instances for Missing Subcircuits ................... 475
Adding Instances of T-Cells for Missing Devices .......... 476
   T-Cell Naming and Parameters .............................. 476
Layout Placement ...................................................... 477
Manual Assisted Routing .......................................... 477
   Setting the Active Net ....................................... 479
Configuring the Automatic Router ............................. 479
   Define the Routing Layers .................................. 479
   Router Settings .............................................. 482
   Set Grid Spacing ........................................... 483
      Router Grid Spacing Field ............................... 483
      Router Grid Offset Field ................................ 484
   Wire Width Computation .................................... 485
   Routing Wire End Styles .................................... 486
   Wide Wire Routing .......................................... 486
   Locating Ports ............................................... 487
      Point Ports .............................................. 488
      Line Ports (1D) .......................................... 489
      Box Ports (2D) ........................................... 489
   Obstruction Regions ........................................ 490
   Breakouts and Notches ...................................... 491
   Netlist Format for Assisted Routing ....................... 492
      Implicit Model Definitions in Assisted Routing ....... 493
      Remapping Model Names ................................. 493
      Alternative Netlist Format .............................. 493
Running the Automatic Router ................................ 494
   Using the SDL Navigator Checklist ........................ 494
Standard Cell Placement and Routing ....................... 495
   Import LEF Technology .................................... 495
   Confirm Routing Layer and Via Definitions ............. 496
   Import LEF Cells ........................................... 497
      Read or Write LEF/DEF Files ............................ 498
   Configure the Placer ....................................... 500
      Choose a Top-Level Cell and Add Site Cells to Define Spacing 501
      Launch Automatic Placement ............................ 502
      Additional Placer Commands ............................ 503
      Add Clock Net Buffers ................................. 504
      Add Fill ................................................. 505
   Place Power Rails ........................................... 505
      Power rails can be placed on the left, right, or both, as shown below ................................. 506
   Route Clock Nets ............................................ 506
      Route Remaining Nets .................................. 506
Sample Standard Cell Placement and Routing Tool Flow .................. 507
   Library setup, done once per library ..................... 507
   Placement and Routing, done for each logic block .... 507
Checking Connectivity ............................................ 508
   Writing SDF ............................................... 508
   Writing SPEF .............................................. 508

14 Interactive DRC 509
L-Edit 16 User Guide—Contents

(Continued)

Introduction .................................................. 509
Setting Up Interactive DRC Rules ........................ 510
  Width ...................................................... 512
  Spacing ................................................... 512
  Surround .................................................. 513
  Overlap .................................................... 513
  Extension .................................................. 514
Running Interactive DRC .................................. 515
Importing Interactive DRC Rules ......................... 515

15 Node Highlighting ........................................ 517
  Introduction ............................................. 517
  Node Highlighting Setup ................................ 517
    Default Location of Saved Node Highlighting Data ........................................... 519
    The Node Highlighting Toolbar .......................................................... 519
    Using Node Highlighting ............................................. 521

16 Add-Ins .................................................... 524
  Add-In Macros Documented in This Chapter .......... 524
  Add-In Macros Documented in Other Chapters .... 524
  Area Calculator ......................................... 525
  Instance Location Summary ............................. 525
  Mask Bias ................................................ 527
  Mark Cells for Flattening ................................ 528
  Layer Fill ................................................. 529
  Macro ...................................................... 531
  Repeat Macro ............................................ 531

Section 2: Design Verification

17 Introduction to Design Verification .................. 532
  Design Verification in L-Edit .......................... 532

18 DRC Standard Rules ..................................... 533
  Design Rule Sets ....................................... 533
    Setups .................................................. 533
      Copying Setup Information to a New File ........ 533
      Combining Rules from Different Files ........... 533
    Generated Layers ..................................... 534
    Exporting DRC Standard Rules to Calibre Format ........................................... 534
  Running DRC from the Command Line .................. 534
    Creating Precompiled Command Files ................. 534
  Design Rule Types ...................................... 535
    Minimum Width ........................................ 535
    Exact Width .......................................... 536
    Not Exist ............................................. 536
    Spacing ................................................ 536
    Surround .............................................. 536
    Overlap ............................................... 537
    Extension .............................................. 537
# L-Edit 16 User Guide—Contents

## Density
- Rule Exceptions .......................... 538
- Acute Angles ............................ 539
- Flag to Append Special Commands ...... 540
  
  *Standard DRC rule:* ..................... 540
  *Equivalent Calibre format rule:* ...... 541

### Specifying DRC Standard Design Rules
- Geometry Flags .......................... 542

### Optimizing Performance
- Checking Incrementally .................. 545
- Checking Hidden Layers ................ 545
- Disabling Rules ........................ 546

## 19 DRC Setup

### Design Rule Sets
- Setting Up DRC .......................... 547
- Running DRC ............................. 550
- Design Rule Check on a Full Cell ...... 550
- Region-Only Design Rule Check ........ 550
- Single Rule Check from a Command File 551
- DRC Progress ........................... 552
- Notification of DRC Completion ........ 552

### Command File Syntax Checking
- DRC Status ............................. 553

### Excluding Cells from DRC
- Debugging DRC Results with Generated Layers 554

### Command File Examples
- Generating Layers ........................ 555
- Generate Layers directly from a Command File 555

## 20 HiPer Verify: Calibre Command Files

### Input Conventions
- Case Sensitivity ........................ 557
- New Line Insensitivity .................. 557
  
  *Navigation in the Text Editor* ......... 557
- Preprocessor Commands .................. 557
- Comments ............................... 557
- Constraints ............................. 557
- Numeric Expressions .................... 558
- Reserved Symbols ........................ 558
- Reserved Keywords ........................ 559

### Command File Examples
- A Basic Command File ........................ 560

### Command Usage
- Intermediate Layer Rules ................ 562
- Edge Directed Output .................... 562
- Polygon Directed Output ................. 563
- Output Reports .......................... 563

### Summary and Classification of Commands
- Polygon Layer Selectors ................ 563
- Edge Layer Selectors ................... 564
- Layer Constructors ...................... 565
- Omitted Environment Setup Commands ... 565

### Function Overview
- ........................................ 567
L-Edit 16 User Guide—Contents

(Continued)

Environment Setup .................................................. 567
Operating Commands .................................................. 567
Hierarchy Modification Commands .............................. 567
Geometry Flags ......................................................... 567
Drawn Layer Definitions ........................................ 568
Connect and Connectivity Commands ........................ 568
Antenna Rules ......................................................... 570
Polygon Boolean Operations ..................................... 570
Utility Layer Generation Operations ......................... 570
Polygon Size Operations ........................................... 570
Two-Layer Polygon Selection Operations ................... 570
Single-Layer Polygon Selection Operations .................. 571
Polygon Area Operations ......................................... 571
Polygon Edge Operations ......................................... 571
Edge, Length and Angle Operations .......................... 571
Edge Selection Operations ....................................... 571
Dimensional Check Operations .................................. 572
Text Based Operations ............................................. 573
Netlist Extraction Operations ................................... 573

Functions ................................................................. 574

AND ................................................................. 575
ANGLE ............................................................... 576
AREA ................................................................. 577
ATTACH ............................................................... 578
COINCIDENT EDGE .................................................. 579
COINCIDENT INSIDE EDGE ....................................... 580
COINCIDENT OUTSIDE EDGE ................................... 581
CONNECT ............................................................ 582
CONVEX EDGE ....................................................... 583
  Detailed Endpoint Specification ................................ 583
  Edge Layer Input ................................................ 584
COPY ................................................................. 585
CUT ................................................................. 586
DENSITY .............................................................. 587
  If the WINDOW option is not present then: .................. 588
  If the WINDOW option is present: ........................... 589
DEVICE ............................................................... 590
DISCONNECT ........................................................ 595
DMACRO and CMACRO ............................................. 596
  Defining Macros—DMACRO ................................... 596
  Invoking Macros—CMACRO ................................... 596
Rule Files ............................................................. 596
DONUT ............................................................... 599
DRAWN ACUTE ....................................................... 600
DRAWN OFFGRID .................................................... 601
DRAWN SKEW ........................................................ 602
DRC INCREMENTAL CONNECT .................................. 603
DRC MAXIMUM RESULTS ......................................... 604
DRC PRINT AREA ..................................................... 605
DRC PRINT PERIMETER ............................................ 606
DRC SELECT CHECK ............................................... 607
DRC TOLERANCE FACTOR ......................................... 608
DRC UNSELECT CHECK ............................................. 609
ENC ................................................................. 610
ENCLOSE ............................................................. 615
ENCLOSE RECTANGLE .............................................. 616
EXCLUDE CELL ....................................................... 617
EXPAND CELL ........................................................ 618
L-Edit 16 User Guide—Contents

(Continued)

TEXT DEPTH .......................................................... 693
TEXT LAYER .......................................................... 694
TITLE ................................................................. 695
TOUCH ............................................................... 696
TOUCH EDGE ......................................................... 697
TOUCH INSIDE EDGE ............................................. 698
TOUCH OUTSIDE EDGE ......................................... 699
VARIABLE ......................................................... 700
VERTEX ............................................................ 701
VIRTUAL CONNECT COLON ....................................... 702
VIRTUAL CONNECT NAME ....................................... 703
VIRTUAL CONNECT SEMICOLON AS COLON ............... 704
WITH EDGE ...................................................... 705
WITH NEIGHBOR .................................................. 706
WITH TEXT ...................................................... 707
WITH WIDTH ..................................................... 708
XOR ................................................................. 709

Supported Calibre Commands ............................. 710
Unsupported Options ........................................... 712
Unsupported Calibre DRC/Extraction Commands ...... 713
Unsupported Database Commands ......................... 714
Unsupported Database Classification Commands ...... 715
 Unsupported Design for Manufacturing Commands ... 716
Unsupported Electrical Rule Check Commands ....... 717
Unsupported Inductance Commands ...................... 718
Unsupported Litho Commands ............................... 719
Unsupported LVS Commands ................................. 720
Unsupported Mask Data Prep Commands ................. 721
Unsupported Parasitic Extraction Commands .......... 722

21 HiPer Verify: Dracula Command Files 723

Introduction ...................................................... 723
Structure of a Dracula File ..................................... 723
Command Usage ................................................. 723
Conjunctive Rules .............................................. 724
Navigation in the Text Editor ............................... 724
Attaching Text .................................................. 725
Simple Example of a Dracula File ......................... 726

Description Block Commands ............................... 727
DELCEL .......................................................... 728
FLAG-ACUTEANGLE ........................................... 729
FLAG-NON45 ................................................... 730
FLAG-OFFGRID/FLAG-PTH-OFFGRID ................. 731
FLAG-SELFINTERS/FLAG-SELFTOUCH ............... 732
RESOLUTION .................................................. 733
SCALE ........................................................ 734
TEXT-LEVEL .................................................. 735
TEXT-PRI-ONLY ............................................... 736

Input-Layer Block Commands .............................. 736
Layer Assignment ............................................. 737
Layer-Name Definition ....................................... 739
CONNECT LAYER .............................................. 742
IDTEXT ........................................................ 743
TEXTSEQUENCE ............................................... 744
L-Edit 16 User Guide—Contents

(Continued)

Operation Block Commands ........................................... 744
   AND ......................................................... 745
   ANDNOT ............................................... 746
   AREA ..................................................... 747
   CAT ....................................................... 748
   CONNECT ............................................... 749
   CORNER ............................................... 750
   COVERAGE .............................................. 751
   DEVTAG .................................................. 753

   Circuit Element Extract Commands .................................. 755
   EDGECHK ................................................ 756
   ELEMENT BJT ........................................... 757
   ELEMENT CAP ........................................... 759
   ELEMENT DIO ............................................ 760
   ELEMENT IN? ............................................ 761
   ELEMENT LDD ........................................... 762
   ELEMENT MOS ........................................... 764
   ELEMENT RES ............................................ 766
   ENC .................................................... 768
   EQUATION .............................................. 774
   EXT ..................................................... 776
   GROW .................................................... 781
   HOLE .................................................... 782
   INT ..................................................... 783
   LENGTH .................................................. 787
   LEXTRACT .............................................. 788

   Antenna Checks ............................................... 788
   NOT ...................................................... 791
   OCTBIAS ............................................... 792
   OR ....................................................... 793
   PARSET .................................................. 794

   Inductors .................................................. 795
   PLENGTH ................................................ 796
   RECTCHK ............................................... 797
   SCONNECT .............................................. 798
   SELECT ANGLE ......................................... 799

   SELECT BY LABEL ....................................... 800
   SELECT CONN .......................................... 801
   SELECT CUT, TOUCH, ENCLOSE, OVERLAP ...................... 802
   SELECT INSIDE, OUTSIDE, HOLE .............................. 804
   SELECT LABEL ......................................... 805
   SELECT VERTEX ........................................ 806
   SHRINK .................................................. 807
   SIZE ..................................................... 808

   SNAP ..................................................... 809
   STAMP ................................................... 810
   WIDTH .................................................... 811
   XOR ...................................................... 815

Unsupported Dracula Commands ................................. 816

22 HiPer Verify: Assura Command Files ...................... 823
   Typographic Syntax ..................................... 823
   Terminology ............................................ 823
   Rule Syntax and Formatting .................................. 824
   Structure of an Assura File .................................. 824
# Contents

## Functions By Category

- Layer Definition Rules ............................................. 832
- Layer Derivation Rules .............................................. 824
- DRC Operations ....................................................... 824

## Command Usage .................................................. 825

### How To Run Assura Rules ........................................ 826

- Assura Setup ......................................................... 826
- Setup Assura — avParameters .................................... 827
- Setup Assura — Switches .......................................... 828
- Setup Assura — userVariables ................................... 829
- Setup Assura — techParams ....................................... 830
- Setup Assura — Prolog Text ...................................... 830

## Initializing the Assura Setup ...................................... 830

- Using Virtuoso Import for Assura Setup ......................... 830
- Parsing the File to Setup Assura ............................... 831

## Functions .......................................................... 832

- Layer Definition Rules ............................................. 832
- Layer Derivation Rules .............................................. 832
- Layer Derivation Rules Using Shape Selection .................. 832
- Layer Derivation Rules using Shape Sizing .................... 833
- Layer Derivation Rules using Shape Generation ............... 834
- Layout Display Rules ............................................... 834
- Layout Extraction Rules ............................................ 834
- Device Extraction Rules ............................................ 834
- DRC Operations ...................................................... 834
- DRC Check Modifiers ............................................... 834
- DRC Output Modifiers ............................................... 835

### DRC Check Keywords ............................................. 840

- area ................................................................. 840
- coverage ............................................................ 840
- enc ................................................................. 841
- notch ............................................................... 841
- ovlp ............................................................... 842
- sep ................................................................. 842
- sepNotch .......................................................... 842
- venc ............................................................... 843
- width ............................................................... 843

### DRC Check Modifiers ............................................. 845

- app ................................................................. 845
- diffNet ............................................................. 846
- edge, edgea, edgeb ............................................... 846
- fig, figa, figb .................................................... 846
- length, lengtha, lengthb ....................................... 846
- lessShielded ....................................................... 847
- notParallel ....................................................... 847
- onlyPerp ........................................................... 847
- opposite ............................................................ 847
- parallel ............................................................ 848
- sameNet .......................................................... 848
- shielded ............................................................ 848
- withPerp ........................................................... 849
- errorLayer ........................................................ 850
- extractCAP ......................................................... 851
- geomAnd .......................................................... 852
- geomAndNot ....................................................... 853

(Continued)
23 Layout vs. Layout 904
Select Cells and Layers to Compare .......................... 904
View LVL Results .................................................. 905
LVL Log File ......................................................... 906
LVL_Results TDB File .............................................. 907
Layout vs. Layout Example ................................. 908

24 Extracting Layout 912
L-Edit 16 User Guide—Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clearing Error Markers</td>
<td>949</td>
</tr>
<tr>
<td>Error Display Options</td>
<td>950</td>
</tr>
<tr>
<td>Viewing Options</td>
<td>950</td>
</tr>
<tr>
<td>Sorting Options</td>
<td>951</td>
</tr>
<tr>
<td><strong>DRC Report Files</strong></td>
<td>951</td>
</tr>
<tr>
<td>DRC Summary Report</td>
<td>951</td>
</tr>
<tr>
<td>DRC Runtime Statistics Report</td>
<td>953</td>
</tr>
<tr>
<td>Exporting a Text File</td>
<td>954</td>
</tr>
<tr>
<td>Displaying Calibre® DRC Results</td>
<td>954</td>
</tr>
<tr>
<td>Finding Open Circuits and Short Circuits</td>
<td>954</td>
</tr>
<tr>
<td><strong>Extract Report Files</strong></td>
<td>955</td>
</tr>
<tr>
<td>Extract Summary Report</td>
<td>955</td>
</tr>
<tr>
<td>Extract Runtime Statistics Report</td>
<td>958</td>
</tr>
<tr>
<td>Tanner “EVI” for Calibre Results Viewing Environment (RVE)</td>
<td>959</td>
</tr>
<tr>
<td>Installing and Configuring EVI</td>
<td>959</td>
</tr>
<tr>
<td>Starting Jobs on the Remote Machine</td>
<td>960</td>
</tr>
<tr>
<td>Setup EVI</td>
<td>960</td>
</tr>
<tr>
<td>Setup Calibre RVE</td>
<td>961</td>
</tr>
<tr>
<td>DRC Results Viewing</td>
<td>963</td>
</tr>
<tr>
<td>Installing an X server to set the Xming Access Controls Option</td>
<td>963</td>
</tr>
<tr>
<td>Installing a Telnet/SSH client to Enable X11 Forwarding</td>
<td>964</td>
</tr>
<tr>
<td><strong>EVI for DRC and LVS</strong></td>
<td>965</td>
</tr>
<tr>
<td>Viewing DRC Errors</td>
<td>965</td>
</tr>
<tr>
<td>Cross Probing</td>
<td>965</td>
</tr>
<tr>
<td>Shorts and Open Circuits</td>
<td>965</td>
</tr>
<tr>
<td>Finding Parasitics</td>
<td>965</td>
</tr>
</tbody>
</table>

27 Cross Probing

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Features</td>
<td>966</td>
</tr>
<tr>
<td>Using Cross Probing</td>
<td>966</td>
</tr>
</tbody>
</table>

28 Introduction to LVS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVS Features</td>
<td>971</td>
</tr>
<tr>
<td><strong>Launching LVS</strong></td>
<td>971</td>
</tr>
<tr>
<td><strong>Input and Output Files</strong></td>
<td>972</td>
</tr>
<tr>
<td>Prematch Files</td>
<td>972</td>
</tr>
<tr>
<td>Syntax</td>
<td>972</td>
</tr>
<tr>
<td>Element Description Files</td>
<td>972</td>
</tr>
<tr>
<td>Syntax</td>
<td>972</td>
</tr>
<tr>
<td>Permutability Statements</td>
<td>973</td>
</tr>
<tr>
<td>Element Description Examples</td>
<td>973</td>
</tr>
<tr>
<td>File Locking</td>
<td>973</td>
</tr>
<tr>
<td>Backup Files</td>
<td>974</td>
</tr>
<tr>
<td><strong>User Interface</strong></td>
<td>974</td>
</tr>
<tr>
<td>Menus</td>
<td>975</td>
</tr>
<tr>
<td>Toolbar</td>
<td>975</td>
</tr>
<tr>
<td>Status Bar</td>
<td>976</td>
</tr>
<tr>
<td><strong>Setup Window</strong></td>
<td>976</td>
</tr>
<tr>
<td>Setup—Input</td>
<td>977</td>
</tr>
<tr>
<td>Setup—Output</td>
<td>978</td>
</tr>
<tr>
<td>Setup—Device Parameters</td>
<td>980</td>
</tr>
<tr>
<td>Setup—Merge Devices</td>
<td>982</td>
</tr>
<tr>
<td>Devices That Can Be Merged</td>
<td>982</td>
</tr>
<tr>
<td>Merging Series MOSFETS</td>
<td>983</td>
</tr>
</tbody>
</table>
L-Edit 16 User Guide—Contents

(Continued)

Resolving Automorph Classes ........................................... 1025
Preiteration Matching ......................................................... 1025
Detailed Trial Matching ....................................................... 1025
Parameter Matching .......................................................... 1026

Permuted Classes in Digital Designs ................................. 1026
Avoiding Permuted Classes .................................................. 1027
LVS Algorithms and Limitations ............................................ 1027
Resolving Discrepancies .................................................... 1028

31  LVS Command-Line Syntax ............................................. 1030
Running LVS from the Command Prompt ............................ 1030
Batch-File Syntax ............................................................. 1030
Running a Batch File ......................................................... 1030
Options ................................................................. 1031
Ignore Bulk Nodes (-b) ....................................................... 1031
Consider Parameters (-cnnnn) ............................................. 1031
Maximum Value Difference (-dv n) ..................................... 1032
Maximum Geometrical Difference (-dg n) ............................ 1032
Element Description File (-e "file") ................................... 1032
Output File Display Options (-f [fapr]) ............................... 1032
Granularity (-%g=n) .......................................................... 1033
Flattened Schematic Netlist (-h "file") ................................. 1033
Fast Iteration (-i) .............................................................. 1034
Delete Disconnected Devices (-k) ........................................ 1034
List Elements and Nodes (-l "file") ..................................... 1034
Merge Devices (-mdevice {ALL | model_list}) ....................... 1034
  Merging Nonpolarized Devices ......................................... 1035
Nonpolarized Elements (-n[rcl]) ......................................... 1036
Output file (-o "file") ...................................................... 1036
Prematch File (-p "file") .................................................... 1037
Input SPICE Syntax (-pspice, -phspice, -hspice) .................. 1037
Merge Series MOSFETs (-r {ALL | model_list}) .................... 1037
Remove Parasitics (-s test=value) ..................................... 1038
Flattened Layout Netlist (-t "file") .................................... 1038
Remove Device Models (-u /model1//model2//.../) ................ 1039
Screen Display Options (-v [fpar]) ..................................... 1039
Delete Shorted Devices (-x) ............................................... 1040
Yes to All Questions (-y [12]) .......................................... 1040
Short Out Device Models (-z /model1//model2//.../) ................ 1040

32  LVS Glossary ........................................................... 1041

Section 3: User-Programmable Interface

33  Introduction to Programming the User Interface ............... 1043
Using Macros-The Basics .................................................. 1043
Using UPI Macros-The Basics ............................................ 1044
  Format for a Compiled (.dll) Macro ................................ 1045
Writing a Macro .............................................................. 1046
  Example 1: Hello World ................................................ 1046
L-Edit 16 User Guide—Contents

(Continued)

Format Outline .................................................. 1046
Defining the Macro Module ................................. 1047
Displaying a Message Box ................................. 1047
Registering the Function as a Macro ......................... 1047
Debugging Macros ............................................. 1048
Binding Macros to Hot Keys .................................. 1049
Binding Macros to Menu Items ......................... 1049
Checking for Cell Locks and Reservations ................. 1049
Example 2: Draw Box ......................................... 1049

How to compile UPI with Visual Studio 2005 .............. 1050
Using a NON-MFC UPI macro in Microsoft Visual Studio 2005 ... 1051
Example 3: Layout Palette .................................... 1060
Creating Resources ........................................... 1061
UPI_Entry_Point() Function ................................. 1062
Displaying and Managing the Palette ......................... 1062
Macro Definitions ............................................. 1064

Copy-Protecting Macro DLLs ................................. 1064
Opening a Copy-Protected DLL ............................. 1065
Creating a Copy-Protected DLL ............................ 1065
Initiating Password Verification .......................... 1065
Verifying the Password ..................................... 1066
Additional Support Routines ................................ 1067

Loading a Macro ............................................... 1068
Entering Library References ................................ 1070
UPI Include Files ............................................. 1071

Running a Macro ............................................... 1071

34 UPI Functions Reference ................................. 1072
Function Categories ........................................... 1072
Data Relationships ........................................... 1072
Global Numerical Limits .................................... 1073
Obsolete Functions ........................................... 1074

Interface Functions ........................................... 1075
Dialog Functions .............................................. 1076
Dialog Button Returns ...................................... 1076
LDIALOG_MsgBox ............................................. 1077
LDIALOG_MultiLineMsgBox ................................. 1078
LDIALOG_AlertBox ........................................... 1079
LDIALOG_YesNoBox .......................................... 1080
LDIALOG_InputBox ........................................... 1081
LDIALOG_MultiLineInputBox ............................... 1082
LDIALOG_PickList ............................................ 1083
LDIALOG_File .................................................. 1084

Cursor and Display Functions ............................... 1088
LCursorGetPosition .......................................... 1089
LCursorGetPositionEx99 ..................................... 1090
LCursorGetSnappedPosition ............................... 1091
LDISPLAY_REFRESH .......................................... 1092
LSTATUSBAR_SETMSG ..................................... 1093
LCELL_HOMEVIEW ........................................... 1094
LCELL_GETVISIBLE ......................................... 1095
LCELL_GETLASTVISIBLE ..................................... 1096
LCELL_MAKEVISIBLE ....................................... 1097
LCELL_MAKEVISIBLENOREFRESH ............................ 1098

UPI Macro Functions .......................................... 1099

L-Edit 16 User Guide
LMacro_Register .......................... 1100
LMacro_BindToHotKey ........................ 1101
LMacro_BindToMenu .......................... 1102
LMacro_BindToMenuAndHotKey_v9_30 .......... 1103
LMacro_IsLoaded ............................ 1108
LMacro_Load ................................ 1109
LMacro_LoadEx1200 .................................. 1110
LMacro_UnLoad ................................ 1111
LMacro_GetNewTCell ......................... 1112

UPI Functions ................................ 1113
LUpi_GetSerialNumber ....................... 1114
LUPI_LogMessage ............................ 1115
LUpi_SetQuietMode ......................... 1116
LUpi_InQuietMode ............................ 1117
LUpi_SetSelectionTool ...................... 1118
LUpi_SetDrawingTool ....................... 1119
LUpi_InsertMenuItemSeparator ............. 1120
LUpi_SetReturnCode ....................... 1121
LUpi_GetReturnCode ....................... 1122
LUpi_SetUpdateDisplayMode ............... 1124
LUpi_GetUpdateDisplayMode ............... 1125
LFormat .................................. 1126
LFormatV .................................. 1127

Windows Functions .......................... 1128
LWindow_GetVisible ...................... 1129
LWindow_GetList ............................ 1130
LWindow_GetNext ......................... 1131
LWindow_IsLast ............................ 1132
LWindow_MakeVisible ...................... 1133
LWindow_Close ............................ 1134
LWindow_CloseAll ......................... 1135
LWindow_EditInPlacePushIn ............... 1136
LWindow_EditInPlacePopOut ............... 1137
LWindow_EditInPlacePopToTop ............. 1138
LWindow_GetType ......................... 1139
LWindow_GetFile ............................ 1140
LWindow_GetCell ............................ 1141
LWindow_GetEditTransform ................. 1142
LWindow_GetTopCell ...................... 1143
LWindow_GetParameters .................... 1144
LWindow_GetWindowHandle .................. 1145
LWindow_NewTextWindow ................... 1146
LWindow_LoadTextFile ..................... 1147
LWindow_SaveToFile ....................... 1148
LWindow_GetText ............................ 1149
LWindow_SetText ............................ 1150
LWindowGetName ......................... 1151
LWindow_SetName ......................... 1152

Database Functions .......................... 1153

Application Functions ........................ 1154
LApp_GetCacheInstances .................. 1155
LApp_GetCacheInstancesSmallerThanNumOfPixels 1156
LApp_GetFillObjectsDuringDrawing ....... 1157
LApp_GetHideInstanceInsidesIfLessThanNumOfPixels 1158
LApp_GetHideObjectsSmallerThanNumOfPixels 1160
LApp_GetHideSmallInstanceInsides .......................... 1161
LApp_GetHideSmallObjects .................................. 1162
LApp_GetInterruptableRendering ............................... 1163
LApp_GetRedrawAllWindows .................................. 1164
LApp_GetVersion ............................................. 1165
LApp_GetVersionDateTime .................................... 1166
LApp_GetFullVersion ......................................... 1167
LApp_GetShowDesignWhileRendering ......................... 1168
LApp_GetShowDesignFirstTimeIncrement ..................... 1169
LApp_GetShowDesignNextTimeIncrement ...................... 1170
LApp_GetRenderingUseCPUForColorMixing .................... 1171
LApp_GetRenderingUseMMX .................................. 1172
LApp_GetRenderingUsePatBltForPatterns ..................... 1173
LApp_GetAllowSelectionOnLockedLayers ..................... 1174
LApp_SetCacheInstances ..................................... 1175
LApp_SetCacheInstancesSmallerThanNumOfPixels ............ 1176
LApp_SetExportMaskDataExportHiddenObjects ............... 1177
LApp_SetFillObjectsDuringDrawing ......................... 1178
LApp_SetHideInstanceInsidesIfLessThanNumOfPixels ........ 1179
LApp_SetHideObjectsSmallerThanNumOfPixels ............... 1181
LApp_SetHideSmallInstanceInsides ......................... 1182
LApp_SetHideSmallObjects .................................. 1183
LApp_SetInterruptableRendering ............................. 1184
LApp_SetRedrawAllWindows .................................. 1185
LApp_SetShowDesignWhileRendering ......................... 1186
LApp_SetShowDesignTimeIncrement ......................... 1187
LApp_SetRenderingUseCPUForColorMixing .................... 1188
LApp_SetRenderingUseMMX .................................. 1189
LApp_SetRenderingUsePatBltForPatterns ..................... 1190
LApp_SetAllowSelectionOnLockedLayers ..................... 1191
LApp.ExitAfterCompletion ................................... 1192

File Functions .................................................. 1193
LFile_New ...................................................... 1195
LFile_Open ..................................................... 1196
LFile_OpenCell ............................................... 1197
LFile_Save ..................................................... 1198
LFile_SaveAs .................................................. 1199
LFile_Close .................................................... 1200
LFile_Find ..................................................... 1201
LFile_GetList .................................................. 1202
LFile_GetNext .................................................. 1203
LFile_GetLock .................................................. 1204
LFile_SetLock .................................................. 1205
LFile_IsChanged ............................................... 1206
LFile_SetChanged ............................................. 1207
LFileGetName .................................................. 1208
LFile_GetAuthor .............................................. 1209
LFile_SetAuthor .............................................. 1210
LFile_GetFabricationCell .................................... 1211
LFile_SetFabricationCell .................................... 1212
LFile_GetOrganization ....................................... 1213
LFile_SetOrganization ....................................... 1214
LFile_GetLayoutVersion ...................................... 1215
LFile_SetLayoutVersion ...................................... 1216
LFile_GetSetupVersion ....................................... 1217
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFile_SetSetupVersion</td>
<td>1218</td>
</tr>
<tr>
<td>LFile_GetInfoText</td>
<td>1219</td>
</tr>
<tr>
<td>LFile_SetInfoText</td>
<td>1220</td>
</tr>
<tr>
<td>LFile_GetEnvironment</td>
<td>1221</td>
</tr>
<tr>
<td>LFile_SetEnvironment</td>
<td>1222</td>
</tr>
<tr>
<td>LFile_GetGrid</td>
<td>1223</td>
</tr>
<tr>
<td>LFile_GetGridEx840</td>
<td>1224</td>
</tr>
<tr>
<td>LFile_GetGrid_v10_00</td>
<td>1225</td>
</tr>
<tr>
<td>LFile_SetGrid_v10_00</td>
<td>1226</td>
</tr>
<tr>
<td>LFile_SetGrid</td>
<td>1227</td>
</tr>
<tr>
<td>LFile_SetGridEx840</td>
<td>1228</td>
</tr>
<tr>
<td>LFile_GetCurveSetup</td>
<td>1229</td>
</tr>
<tr>
<td>LFile_SetCurveSetup</td>
<td>1230</td>
</tr>
<tr>
<td>LFile_GetSelectionParam</td>
<td>1231</td>
</tr>
<tr>
<td>LFile_SetSelectionParam</td>
<td>1232</td>
</tr>
<tr>
<td>LFile_GetUserData</td>
<td>1233</td>
</tr>
<tr>
<td>LFile_SetUserData</td>
<td>1234</td>
</tr>
<tr>
<td>LFile_DeleteUserData</td>
<td>1235</td>
</tr>
<tr>
<td>LFile_ClearUserData</td>
<td>1236</td>
</tr>
<tr>
<td>LFile_DisplayCellBrowser</td>
<td>1237</td>
</tr>
<tr>
<td>LFile_SetLastCurrent</td>
<td>1238</td>
</tr>
<tr>
<td>LFile_GetDesignRuleFlags</td>
<td>1239</td>
</tr>
<tr>
<td>LFile_SetDesignRuleFlags</td>
<td>1240</td>
</tr>
<tr>
<td>LFile_GetResolvedFileName</td>
<td>1241</td>
</tr>
<tr>
<td>LFile_GetVisible</td>
<td>1242</td>
</tr>
<tr>
<td>LFile_IntUtoLocU</td>
<td>1243</td>
</tr>
<tr>
<td>LFile_LocUtoIntU</td>
<td>1244</td>
</tr>
<tr>
<td>LFile_GetDisplayUnitInfo</td>
<td>1245</td>
</tr>
<tr>
<td>LFile_SetDisplayUnit</td>
<td>1246</td>
</tr>
<tr>
<td>LFile_IntUtoIntU</td>
<td>1247</td>
</tr>
<tr>
<td>LFile_DispUtoIntU</td>
<td>1248</td>
</tr>
<tr>
<td>LFile_IntUtoMicrons</td>
<td>1249</td>
</tr>
<tr>
<td>LFile_MicronsToIntU</td>
<td>1250</td>
</tr>
<tr>
<td>LFile_GetCurrentLayerPalette</td>
<td>1251</td>
</tr>
<tr>
<td>LFile_SetCurrentLayerPalette</td>
<td>1252</td>
</tr>
<tr>
<td>LFile_GetNextLayerPalette</td>
<td>1253</td>
</tr>
<tr>
<td>LFile_DefineLayerPalette</td>
<td>1254</td>
</tr>
<tr>
<td>LFile_GetTopLevelLibraryName</td>
<td>1255</td>
</tr>
</tbody>
</table>

**Cell Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCell_New</td>
<td>1256</td>
</tr>
<tr>
<td>LCell_Delete</td>
<td>1257</td>
</tr>
<tr>
<td>LCell_Copy</td>
<td>1258</td>
</tr>
<tr>
<td>LCell_Find</td>
<td>1259</td>
</tr>
<tr>
<td>LCell_FindEx</td>
<td>1260</td>
</tr>
<tr>
<td>LCell_GetFile</td>
<td>1261</td>
</tr>
<tr>
<td>LCell_GetList</td>
<td>1262</td>
</tr>
<tr>
<td>LCell_GetNext</td>
<td>1263</td>
</tr>
<tr>
<td>LCell_GetLock</td>
<td>1264</td>
</tr>
<tr>
<td>LCell_SetLock</td>
<td>1265</td>
</tr>
<tr>
<td>LCell_GetName</td>
<td>1266</td>
</tr>
<tr>
<td>LCellSetName</td>
<td>1267</td>
</tr>
<tr>
<td>LCell_GetAuthor</td>
<td>1268</td>
</tr>
<tr>
<td>LCell_SetAuthor</td>
<td>1269</td>
</tr>
<tr>
<td>LCell_GetOrganization</td>
<td>1270</td>
</tr>
<tr>
<td>LCell_SetOrganization</td>
<td>1271</td>
</tr>
</tbody>
</table>
LCell_GetInfoText ................................. 1277
LCell_SetInfoText ................................. 1278
LCell_MakeLogo ................................. 1279
LCell_GetVersion ............................... 1281
LCell_SetVersion .............................. 1282
LCell_GetCreatedTime ....................... 1283
LCell_GetModifiedTime ...................... 1284
LCell_IsChanged .............................. 1285
LCell_GetView ................................ 1286
LCell_SetView ................................ 1287
LCell_GetMbb .................................. 1288
LCell_GetMbbAll ................................ 1289
LCell_Flatten .................................. 1290
LCell_ClearUserData ......................... 1291
LCell_GetUserData ......................... 1292
LCell_SetUserData ......................... 1293
LCell_DeleteUserData ....................... 1294
LCell_GenerateLayersEx830 ............... 1295
LCell_GenerateLayersEx99 .................. 1297
LCell_GenerateLayers_v10_00 ............. 1298
LCell_GenerateLayers_v11_10 ............. 1300
LCell_SetChanged ............................. 1301
LCell_RunDRC ................................ 1302
LCell_RunDRCEx00 ......................... 1303
LCell_RunDRCEx01 ......................... 1305
LCell_RunDRCCommandFile ................ 1307
LCell_ClearUndoLists ........................ 1308
LCell_GetParameter .......................... 1309
LCell_GetTCellPreviousValue ............. 1310
LCell_SetShowInLists ...................... 1312
LCell_GetShowInLists ...................... 1313
LCell_CalcMBB ................................ 1314
LCell_AddMarker .............................. 1315
LCell_RemoveMarker ......................... 1316
LCell_RemoveAllMarkers .................... 1317
LCell_BooleanOperation .................... 1318
LCell_Slice ................................... 1321
LCell_SetBasePoint ......................... 1322
LCell_GetBasePoint ......................... 1323

Instance Functions .............................. 1324
LInstance_New ................................ 1325
LInstance_New_Ex99 ......................... 1326
LInstance_Delete ......................... 1327
LInstance_Set ...................... 1328
LInstance_Set_Ex99 ...................... 1329
LInstance_Find ................................ 1330
LInstance_FindNext ....................... 1331
LInstance_GetList ....................... 1332
LInstance_GetNext ....................... 1333
LInstance_GetName ....................... 1334
LInstance_SetName ....................... 1335
LInstance_GetCell ....................... 1336
LInstance_GetTransform ................ 1337
LInstance_GetTransform_Ex99 ............ 1338
LInstance_GetRepeatCount ................ 1339
L-Edit 16 User Guide—Contents

(Continued)

**Entity Functions**

- LInstance_GetDelta ........................................ 1340
- LInstance_GetMbb ........................................... 1341
- LInstance_Generate ........................................ 1342
- LInstance_GenerateV ....................................... 1343

**Entity Functions**

- LEntity_PropertyExists .................................... 1345
- LEntity_GetPropertyType ................................... 1347
- LEntity_GetPropertyValueSize ......................... 1348
- LEntity_GetPropertyValue ................................. 1349
- LEntity_AssignProperty .................................. 1350
- LEntity_AssignBlobProperty .............................. 1351
- LEntity_DeleteProperty .................................. 1352
- LEntity_DeleteAllProperties ............................. 1353
- LEntity_CopyAllProperties ............................... 1354
- LEntity_GetFirstProperty ................................ 1355
- LEntity_GetNextProperty ................................ 1356
- LEntity_SetCurrentProperty ................................ 1357
- LEntity_BrowseProperties ................................ 1358
- LEntity_LoadBlobProperty ................................ 1359
- LEntity_SaveBlobProperty ................................ 1360
- LEntity_ReadPropertiesFromFile ...................... 1361
- LEntity_StringToValidPropertyName ................... 1363
- LEntity_ValidPropertyNameToString .................... 1365
- LEntity_WritePropertiesToFile ......................... 1367
- LEntity_StoreAsCompressedBlob ....................... 1369
- LEntity_DecompressBlobToFile ......................... 1370

**Object Functions**

-LObject_Delete ............................................. 1372
-LObject_GetList ........................................... 1373
-LObject_GetNext .......................................... 1374
-LObject_Transform ......................................... 1375
-LObject_Transform_Ex99 ................................... 1376
-LOBject.GetInstance ........................................ 1377
-LOBject_GetMbb ........................................... 1378
-LOBject_GetShape .......................................... 1379
-LOBject_GetGeometry ...................................... 1380
-LOBject_GetVertexList .................................... 1381
-LOBject_GetLayer .......................................... 1382
-LOBject_Area .............................................. 1383
-LOBject_Perimeter ........................................ 1384
-LOBject_GetLayer .......................................... 1385
-LOBject_GetGDSIIDataType .............................. 1386
-LOBject_SetGDSIIDataType ................................ 1387
-LOBject_ChangeLayer ...................................... 1388
-LOBject_ConvertToPolygon ............................... 1389
-LOBject_Copy .............................................. 1390
-LOBject_SnapToGrid ....................................... 1391
-LOBject_SnapToMfgGrid ................................... 1392
-LOBject_DistanceToPoint ................................. 1393

**Vertex Functions**

- LVertex_GetCount .......................................... 1396
- LVertex_GetArray .......................................... 1397
- LVertex_GetNext .......................................... 1398
- LVertex_GetNext .......................................... 1399
- LVertex_GetPoint ......................................... 1400
- LVertex_SetPoint ......................................... 1401
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVertex_Add</td>
<td>1402</td>
</tr>
<tr>
<td>LVertex_Delete</td>
<td>1403</td>
</tr>
<tr>
<td>LVertex_AddCurve</td>
<td>1404</td>
</tr>
<tr>
<td>LVertex_GetCurve</td>
<td>1405</td>
</tr>
<tr>
<td>LVertex_GetCurveEX</td>
<td>1406</td>
</tr>
<tr>
<td>LVertex_GetCurveExactCenter</td>
<td>1407</td>
</tr>
<tr>
<td>LVertex_HasCurve</td>
<td>1408</td>
</tr>
<tr>
<td>LVertex_SetCurve</td>
<td>1409</td>
</tr>
<tr>
<td>LVertex_RemoveCurve</td>
<td>1410</td>
</tr>
<tr>
<td><strong>Box Functions</strong></td>
<td></td>
</tr>
<tr>
<td>LBox_New</td>
<td>1411</td>
</tr>
<tr>
<td>LBox_Set</td>
<td>1412</td>
</tr>
<tr>
<td>LBox_GetRect</td>
<td>1413</td>
</tr>
<tr>
<td>LBox_GetRect</td>
<td>1414</td>
</tr>
<tr>
<td><strong>Circle Functions</strong></td>
<td></td>
</tr>
<tr>
<td>LCircle_New</td>
<td>1415</td>
</tr>
<tr>
<td>LCircle_Set</td>
<td>1416</td>
</tr>
<tr>
<td>LCircle_GetCenter</td>
<td>1417</td>
</tr>
<tr>
<td>LCircle_GetRadius</td>
<td>1418</td>
</tr>
<tr>
<td>LCircle_GetRect</td>
<td>1419</td>
</tr>
<tr>
<td><strong>Torus and Pie Functions</strong></td>
<td></td>
</tr>
<tr>
<td>LPie_CreateNew</td>
<td>1421</td>
</tr>
<tr>
<td>LPie_GetParams</td>
<td>1422</td>
</tr>
<tr>
<td>LPie_SetParams</td>
<td>1424</td>
</tr>
<tr>
<td>LTorus_CreateNew</td>
<td>1425</td>
</tr>
<tr>
<td>LTorus_GetParams</td>
<td>1427</td>
</tr>
<tr>
<td>LTorus_SetParams</td>
<td>1429</td>
</tr>
<tr>
<td><strong>Wire Functions</strong></td>
<td></td>
</tr>
<tr>
<td>LWire_New</td>
<td>1432</td>
</tr>
<tr>
<td>LWire_GetWidth</td>
<td>1433</td>
</tr>
<tr>
<td>LWire_GetCapType</td>
<td>1434</td>
</tr>
<tr>
<td>LWire_GetJoinType</td>
<td>1435</td>
</tr>
<tr>
<td>LWire_GetMiterAngle</td>
<td>1436</td>
</tr>
<tr>
<td>LWire_GetLength</td>
<td>1437</td>
</tr>
<tr>
<td>LWire_GetSquares</td>
<td>1438</td>
</tr>
<tr>
<td>LWire_GetResistance</td>
<td>1439</td>
</tr>
<tr>
<td>LWire_SetWidth</td>
<td>1440</td>
</tr>
<tr>
<td>LWire_SetJoinType</td>
<td>1441</td>
</tr>
<tr>
<td>LWire_SetCapType</td>
<td>1442</td>
</tr>
<tr>
<td>LWire_SetCapType</td>
<td>1443</td>
</tr>
<tr>
<td>LWire_SetMiterAngle</td>
<td>1444</td>
</tr>
<tr>
<td><strong>Polygon Functions</strong></td>
<td></td>
</tr>
<tr>
<td>LPolygon_New</td>
<td>1445</td>
</tr>
<tr>
<td>LPolygon_WireToPolygon</td>
<td>1446</td>
</tr>
<tr>
<td>LPolygon_CircleToPolygon</td>
<td>1447</td>
</tr>
<tr>
<td>LPolygon_HasCurve</td>
<td>1448</td>
</tr>
<tr>
<td>LPolygon_RemoveAllCurves</td>
<td>1449</td>
</tr>
<tr>
<td>LPolygon_StraightenAllCurves</td>
<td>1450</td>
</tr>
<tr>
<td><strong>Port Functions</strong></td>
<td></td>
</tr>
<tr>
<td>LPort_New</td>
<td>1452</td>
</tr>
<tr>
<td>LPort_Delete</td>
<td>1453</td>
</tr>
<tr>
<td>LPort_Find</td>
<td>1454</td>
</tr>
<tr>
<td>LPort_FindNext</td>
<td>1455</td>
</tr>
<tr>
<td>LPort_GetList</td>
<td>1456</td>
</tr>
<tr>
<td>LPort_GetNext</td>
<td>1457</td>
</tr>
<tr>
<td>LPort_GetText</td>
<td>1458</td>
</tr>
<tr>
<td>LPort_GetTextSize</td>
<td>1459</td>
</tr>
<tr>
<td>LPort_SetText</td>
<td>1460</td>
</tr>
<tr>
<td>LPort_GetTextSize</td>
<td>1461</td>
</tr>
</tbody>
</table>
LPort_GetLayer .............................................. 1462
LPort_GetMbb .............................................. 1463
LPort_GetRect ............................................. 1464
LPort_Set .................................................. 1465
LPort_SetTextSize ......................................... 1466
LPort_GetTextLabel ........................................ 1468
LPort_SetTextLabel ....................................... 1470
Port Alignment Definitions ................................. 1471
LPort_GetDefaultTextLabel ............................... 1472
LPort_SetDefaultTextLabel ............................... 1473
LNew ...................................................... 1474
LGetName ................................................... 1475
LSetName ................................................... 1476
LNew_With_Object ......................................... 1477
LGetObjectList ............................................ 1478
LGetObjectsNext .......................................... 1479
LAddObject ............................................... 1480
LRemoveObject ............................................ 1481
LGetPort .................................................. 1482
LIsLegacy ................................................. 1483
LDeletePortOnly .......................................... 1484
LSetTerminalType ......................................... 1485
LGetTerminalType ......................................... 1486
LSetAccessDirection ...................................... 1487
LGetAccessDirection ...................................... 1488
LSetMustConnect ......................................... 1489
LGetPinName .............................................. 1490
LLayer_FindLayerName .................................... 1492
LLayer_FindPurposeName .................................. 1493
Selection Functions ....................................... 1494
LCut ....................................................... 1495
LCopy ....................................................... 1496
LConnectSegments ......................................... 1497
LCopy ....................................................... 1498
LPasteToLayer ............................................. 1499
LClear ..................................................... 1500
LSelectAll ................................................. 1501
LDeselectAll .............................................. 1502
LAddObject ............................................... 1503
LRemoveObject ............................................ 1504
GetObject ................................................. 1505
AddAllObjectsOnLayer ..................................... 1506
RemoveAllObjectsOnLayer ................................. 1507
AddAllObjectsInRect ...................................... 1508
RemoveAllObjectsInRect ................................. 1509
GetList .................................................... 1510
GetNext ..................................................... 1511
GetLayer ................................................... 1512
ChangeLayer .............................................. 1513
Move ....................................................... 1514
Duplicate .................................................. 1515
Group ....................................................... 1516
UnGroup ..................................................... 1517
Flatten ..................................................... 1518
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSelection_Merge</td>
<td>1519</td>
</tr>
<tr>
<td>LSelection_FlipHorizontal</td>
<td>1520</td>
</tr>
<tr>
<td>LSelection_FlipVertical</td>
<td>1521</td>
</tr>
<tr>
<td>LSelection_SliceHorizontal</td>
<td>1522</td>
</tr>
<tr>
<td>LSelection_SliceVertical</td>
<td>1523</td>
</tr>
<tr>
<td>LSelection_Rotate</td>
<td>1524</td>
</tr>
<tr>
<td>LSelection_RotateAroundPoint</td>
<td>1525</td>
</tr>
<tr>
<td>LSelection_SnapToMfgGrid</td>
<td>1527</td>
</tr>
<tr>
<td><strong>Layer Functions</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Design Layer Functions</strong></td>
<td></td>
</tr>
<tr>
<td>LLayer_New</td>
<td>1528</td>
</tr>
<tr>
<td>LLayer_Delete</td>
<td>1529</td>
</tr>
<tr>
<td>LLayer_Find</td>
<td>1530</td>
</tr>
<tr>
<td>LLayer_FindGDS</td>
<td>1531</td>
</tr>
<tr>
<td>LLayer_GetList</td>
<td>1532</td>
</tr>
<tr>
<td>LLayer_GetNext</td>
<td>1533</td>
</tr>
<tr>
<td>LLayer_PrecedingLayer</td>
<td>1534</td>
</tr>
<tr>
<td>LLayer_PrecedingLayerEx99</td>
<td>1535</td>
</tr>
<tr>
<td>LLayer GetName</td>
<td>1536</td>
</tr>
<tr>
<td>LLayer_SetName</td>
<td>1537</td>
</tr>
<tr>
<td>LLayer_GetParameters</td>
<td>1538</td>
</tr>
<tr>
<td>LLayer_GetParametersEx830</td>
<td>1539</td>
</tr>
<tr>
<td>LLayer_GetParametersEx1512</td>
<td>1540</td>
</tr>
<tr>
<td>LLayer_SetParameters</td>
<td>1541</td>
</tr>
<tr>
<td>LLayer_GetName</td>
<td>1542</td>
</tr>
<tr>
<td>LLayer_SetName</td>
<td>1543</td>
</tr>
<tr>
<td>LLayer_GetParametersEx830</td>
<td>1544</td>
</tr>
<tr>
<td>LLayer_GetParametersEx1512</td>
<td>1545</td>
</tr>
<tr>
<td>LLayer_SetParameters</td>
<td>1546</td>
</tr>
<tr>
<td>LLayer_GetCap</td>
<td>1547</td>
</tr>
<tr>
<td>LLayer_GetCap</td>
<td>1548</td>
</tr>
<tr>
<td>LLayer_SetCap</td>
<td>1549</td>
</tr>
<tr>
<td>LLayer_GetRho</td>
<td>1550</td>
</tr>
<tr>
<td>LLayer_GetRho</td>
<td>1551</td>
</tr>
<tr>
<td>LLayer_GetCurrent</td>
<td>1552</td>
</tr>
<tr>
<td>LLayer_SetCurrent</td>
<td>1553</td>
</tr>
<tr>
<td>LLayer_GetSpecial</td>
<td>1554</td>
</tr>
<tr>
<td>LLayer_SetSpecial</td>
<td>1555</td>
</tr>
<tr>
<td>LLayer_MoveLayer</td>
<td>1556</td>
</tr>
<tr>
<td>LLayer_Copy</td>
<td>1557</td>
</tr>
<tr>
<td><strong>Generated Layer Functions</strong></td>
<td></td>
</tr>
<tr>
<td>LLayer_GetDerivedList</td>
<td>1558</td>
</tr>
<tr>
<td>LLayer_GetDerivedNext</td>
<td>1559</td>
</tr>
<tr>
<td>LLayer_IsDerived</td>
<td>1560</td>
</tr>
<tr>
<td>LLayer_EnableAllDerived</td>
<td>1561</td>
</tr>
<tr>
<td>LLayer_DisableAllDerived</td>
<td>1562</td>
</tr>
<tr>
<td>LLayer_GetDerivedParameters</td>
<td>1563</td>
</tr>
<tr>
<td>LLayer_GetDerivedParametersEx830</td>
<td>1564</td>
</tr>
<tr>
<td>LLayer_SetDerivedParameters</td>
<td>1565</td>
</tr>
<tr>
<td>LLayer_SetDerivedParametersEx830</td>
<td>1566</td>
</tr>
<tr>
<td>LLayer_DestroyDerivedParameter</td>
<td>1567</td>
</tr>
<tr>
<td>LLayer_DestroyDerivedParameterEx840</td>
<td>1568</td>
</tr>
<tr>
<td>LCell_GenerateLayers</td>
<td>1569</td>
</tr>
<tr>
<td>LCell_GenerateLayers_v11_10</td>
<td>1570</td>
</tr>
<tr>
<td>LCell_GenerateLayers_v10_00</td>
<td>1571</td>
</tr>
<tr>
<td>LCell_ClearGenerateLayers</td>
<td>1572</td>
</tr>
<tr>
<td><strong>Layer Rendering Functions</strong></td>
<td></td>
</tr>
<tr>
<td>LPass_New</td>
<td>1573</td>
</tr>
<tr>
<td>LPass_GetList</td>
<td>1574</td>
</tr>
<tr>
<td>LPass_GetNext</td>
<td>1575</td>
</tr>
</tbody>
</table>
L-Edit 16 User Guide—Contents

(Continued)

LPass_GetParameters ........................................ 1579
LPass_SetParameters ........................................ 1580
Layer_GetRenderingAttribute ................................. 1581
Layer_SetRenderingAttribute ................................ 1582
Layer_GetRenderingObjectName ............................... 1583
LFile_GetColorPalette ....................................... 1584
LFile_GetColorPaletteNumColors ............................. 1585
LFile_GetColorPaletteSortBy ................................. 1586
LFile_SetColorPalette ....................................... 1587
LFile_SetColorPaletteNumColors ............................. 1588
LFile_SetColorPaletteSortBy ................................. 1589

Technology Setup Functions ................................. 1590
LFile_GetTechnology .......................................... 1591
LFile_SetTechnology ......................................... 1592
LFile_SetTechnologyName .................................... 1593
LFile_SetTechnologyUnitNum ................................ 1594
LFile_SetTechnologyUnitDenom ............................... 1595
LFile_SetTechnologyUnitName ................................ 1596
LFile_SetTechnologyLambdaNum .............................. 1597
LFile_SetTechnologyLambdaDenom ............................ 1598
LFile_GetTechnologyEx840 ................................... 1599
LFile_SetTechnologyEx840 ................................... 1600

Color Palette Functions ...................................... 1601
LFile_GetColorPalette ....................................... 1602
LFile_GetColorPaletteNumColors ............................. 1603
LFile_GetColorPaletteSortBy ................................. 1604
LFile_SetColorPalette ....................................... 1605
LFile_SetColorPaletteNumColors ............................. 1606
LFile_SetColorPaletteSortBy ................................. 1607

Import/Export Functions ..................................... 1608

CIF Setup Functions .......................................... 1609
LFile_GetCIFParameters ..................................... 1610
LFile_SetCIFParameters ..................................... 1611
LFile_ImportCIF ................................................ 1612

GDSII Setup Functions ........................................ 1613
LFile_GetGDSParameters ..................................... 1614
LFile_SetGDSParameters ..................................... 1616
LFile_ImportGDSII ............................................ 1617
LFile_ExportGDSII ............................................ 1618
LFile_ImportTechnology_GDSLayerMap ....................... 1620
LFile_ImportTechnology_LEF ................................ 1621

DRC Functions ................................................ 1622
LDrcRule_Add .................................................. 1623
LDrcRule_Delete .............................................. 1624
LDrcRule_Find ................................................ 1625
LDrcRule_GetList ............................................. 1626
LDrcRule_GetNext ............................................. 1627
LDrcRule_SetRuleSet .......................................... 1628
LDrcRule_SetTolerance ....................................... 1629
LDrcRule_GetParameters ..................................... 1630
LDrcRule_SetParameters ..................................... 1631
LDRCRule_DestroyParameter .................................. 1632
LDRC_Run ....................................................... 1634
LFile_GetBinSize .............................................. 1635
LFile_SetBinSize .............................................. 1636
LFile_GetDrcFlags ............................................ 1637
L-Edit 16 User Guide—Contents

LBooleanOperation .................................................. 1698
LCapType ................................................................. 1699
LCell ................................................................. 1700
LCell_GetParameterAsString .................................... 1701
LCell_GetParameterName ........................................ 1702
LCell_GetParameterType .......................................... 1703
LCIFParam ............................................................ 1704
LOverwriteCellsScopeOnImport .................................. 1705
LColor ................................................................. 1706
LCoord ................................................................. 1707
LCore ................................................................. 1708
LCursorType ........................................................ 1709
LCurve ................................................................. 1710
LDerivedLayerAreaOperation ...................................... 1711
LDerivedLayerBoolOperation ..................................... 1713
LDerivedLayerDensityOperation ............................... 1715
LDerivedLayerParam ............................................... 1716
LDerivedLayerParamEx830 ....................................... 1717
LDerivedLayerSelectOperation .................................. 1719
LDesignRuleFlags .................................................. 1721
LDesignRuleParam ................................................ 1722
LDlgItem ............................................................... 1723
LDisplayUnitParam ................................................ 1724
LDrcFlags ............................................................ 1725
LDrcRule .............................................................. 1726
LDrcRuleType ........................................................ 1727
LDrcStatus ........................................................... 1728
LEntity ................................................................. 1729
LEnvironment ......................................................... 1730
LExtractOptions ..................................................... 1731
LExtractOptionsEx840 ............................................. 1733
LFile ................................................................. 1736
LFileType .............................................................. 1737
LGDSParam .......................................................... 1738
LGDSParamEx ........................................................ 1739
LGDSExport Scope .................................................. 1741
LGDSExportLogParams .............................................. 1745
LGeomType ............................................................ 1746
LGrid ................................................................. 1747
LGridEx840 ........................................................... 1748
LGrid_v10_00 .......................................................... 1750
LInstance ............................................................. 1752
LJoinType ............................................................. 1753
LLabel ................................................................. 1754
LLayer ................................................................. 1755
LLayerParam ........................................................ 1756
LLayerParamEx830 .................................................. 1757
LLayerParamEx1512 ............................................... 1759
LLayer_GetParametersEx1512 .................................. 1760
LLayer_SetParametersEx1512 .................................. 1761
LLayerViewStatus ................................................... 1762
LLen ................................................................. 1763
LMagnification ....................................................... 1764
LMarker ............................................................... 1765
LMarkerParam ....................................................... 1766
L-Edit 16 User Guide—Contents

35 LComp Functions Reference 1811

Introduction ................................................................. 1811
Composition with LComp .............................................. 1811
Initializing LComp ...................................................... 1811
Elements ................................................................. 1812
State Variables ......................................................... 1813

State Functions .......................................................... 1814
LC_SetReferencePoint ................................................. 1815
LC_GetReferencePoint ................................................. 1816
LC_SetAbutmentType .................................................. 1817
LC_GetAbutmentType .................................................. 1818
LC_SetPlacementOrientation ....................................... 1819
LC_GetPlacementOrientation ....................................... 1820

LMarkerParam_V11 .......................................................... 1767
LObject ......................................................................... 1769
LOrientation ................................................................... 1770
LOrientation_Ex99 .......................................................... 1771
LOutlineStyle ............................................................... 1772
LOutlineUnitType .......................................................... 1773
LPalette ......................................................................... 1774
LPass ............................................................................. 1775
LPassMode ................................................................. 1776
LPassParam ............................................................... 1777
LPassType ................................................................. 1778
LPieParams ............................................................... 1779
LPoint ........................................................................... 1780
LPort ............................................................................. 1781
LPropertyType ............................................................. 1782
LRect ............................................................................. 1783
LRenderingAttribute .................................................... 1784
LRenderingAttributeIndex ............................................ 1786
LRenderingMode ........................................................... 1787
LSelection ................................................................. 1788
LSelectionParam .......................................................... 1789
LShapeType ............................................................... 1790
LSpecialLayer ............................................................. 1791
LStatus ......................................................................... 1792
LStipple ........................................................................ 1793
tech_unit_type ............................................................. 1795
LTCellParameterType ..................................................... 1796
LTechnology ............................................................... 1797
LTechnologyEx840 ....................................................... 1798
LTerminalType ............................................................. 1799
Port Access Direction ................................................... 1799
LTorusParams ............................................................. 1800
LTransform ................................................................. 1801
LTransform_Ex99 ............................................................ 1802
LVertex .......................................................................... 1803
LWindow ........................................................................ 1804
LWindowType .............................................................. 1805
LWireConfig ............................................................... 1806
LWireConfigBits .......................................................... 1807
LWireParam ............................................................... 1808
UPIDrawingToolType .................................................... 1809
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC_AddPlacementOrientation</td>
<td>1821</td>
</tr>
<tr>
<td>LC_SubtractPlacementOrientation</td>
<td>1822</td>
</tr>
<tr>
<td>LC_SetCompositionDirection</td>
<td>1823</td>
</tr>
<tr>
<td>LC_GetCompositionDirection</td>
<td>1824</td>
</tr>
<tr>
<td>LC_SetPlacementOverlap</td>
<td>1825</td>
</tr>
<tr>
<td>LC_GetPlacementOverlap</td>
<td>1826</td>
</tr>
<tr>
<td>LC_SetXYPlacementPosition</td>
<td>1827</td>
</tr>
<tr>
<td>LC_GetXYPlacementPosition</td>
<td>1828</td>
</tr>
<tr>
<td>LC_SetXPlacementPosition</td>
<td>1829</td>
</tr>
<tr>
<td>LC_GetXPlacementPosition</td>
<td>1830</td>
</tr>
<tr>
<td>LC_SetYPlacementPosition</td>
<td>1831</td>
</tr>
<tr>
<td>LC_GetYPlacementPosition</td>
<td>1832</td>
</tr>
<tr>
<td>LC_IncrementXPlacementPosition</td>
<td>1833</td>
</tr>
<tr>
<td>LC_IncrementYPlacementPosition</td>
<td>1834</td>
</tr>
</tbody>
</table>

**Placement Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC_Position</td>
<td>1835</td>
</tr>
<tr>
<td>LC_Instance</td>
<td>1836</td>
</tr>
<tr>
<td>LC_Generate</td>
<td>1837</td>
</tr>
<tr>
<td>LC_Align</td>
<td>1838</td>
</tr>
<tr>
<td>LC_InstanceAlign</td>
<td>1839</td>
</tr>
<tr>
<td>LC_GenerateAlign</td>
<td>1840</td>
</tr>
<tr>
<td>LC_CreateAlign</td>
<td>1841</td>
</tr>
</tbody>
</table>

**Position Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC_GetPoint</td>
<td>1842</td>
</tr>
<tr>
<td>LC_GetPointEX</td>
<td>1843</td>
</tr>
<tr>
<td>LC_GetPlacementRect</td>
<td>1844</td>
</tr>
<tr>
<td>LC_GetPlacementRectEX</td>
<td>1845</td>
</tr>
<tr>
<td>LC_GetElementWidth</td>
<td>1846</td>
</tr>
<tr>
<td>LC_GetElementHeight</td>
<td>1847</td>
</tr>
</tbody>
</table>

**Geometry Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC_StartWire</td>
<td>1849</td>
</tr>
<tr>
<td>LC_AddWirePoint</td>
<td>1850</td>
</tr>
<tr>
<td>LC_EndWire</td>
<td>1851</td>
</tr>
<tr>
<td>LC_CreateBox</td>
<td>1852</td>
</tr>
<tr>
<td>LC_CreateCircle</td>
<td>1853</td>
</tr>
<tr>
<td>LC_CreatePort</td>
<td>1854</td>
</tr>
<tr>
<td>LC_StartPolygon</td>
<td>1855</td>
</tr>
<tr>
<td>LC_AddPolygonPoint</td>
<td>1856</td>
</tr>
<tr>
<td>LC_EndPolygon</td>
<td>1857</td>
</tr>
</tbody>
</table>

**Utility Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC_Push</td>
<td>1859</td>
</tr>
<tr>
<td>LC_Pop</td>
<td>1860</td>
</tr>
<tr>
<td>LC_DiskFileExists</td>
<td>1861</td>
</tr>
<tr>
<td>LC_DiskFileDelete</td>
<td>1862</td>
</tr>
<tr>
<td>LC_DiskFileRename</td>
<td>1863</td>
</tr>
<tr>
<td>LC_Lambda</td>
<td>1864</td>
</tr>
<tr>
<td>LC_Microns</td>
<td>1865</td>
</tr>
<tr>
<td>LC_InMicrons</td>
<td>1866</td>
</tr>
<tr>
<td>LC_CellOpen</td>
<td>1867</td>
</tr>
<tr>
<td>LC_CellNew</td>
<td>1868</td>
</tr>
<tr>
<td>LC_CellClose</td>
<td>1869</td>
</tr>
<tr>
<td>LC_CellExists</td>
<td>1870</td>
</tr>
<tr>
<td>LC_PropagatePorts</td>
<td>1871</td>
</tr>
<tr>
<td>LC_Trace</td>
<td>1872</td>
</tr>
<tr>
<td>LC_TraceFile</td>
<td>1873</td>
</tr>
<tr>
<td>LC_PlaceMarkerAtCurrentPos</td>
<td>1874</td>
</tr>
<tr>
<td>LC_PlaceMarkerAtCurrentPosition</td>
<td>1875</td>
</tr>
</tbody>
</table>
L-Edit 16 User Guide—Contents

(Continued)

Typedefs .......................... 1876
AbutTo ............................ 1877
CompositionDirectionType ............ 1878
RelativeTo .......................... 1879

Examples .......................... 1880
Logo Generator ...................... 1881
Buffer .............................. 1883
Matched Dual Capacitor Array ......... 1885
Decoder ............................. 1892

36 Alphabetical List of UPI and LComp Functions 1898

Appendix: Legacy Tools

A1 Introduction to Placement and Routing 1914
   Placement and Routing in L-Edit .................. 1914
   Standard Cell Place and Route (SPR) ............. 1914

A2 Placing and Routing Standard Cells 1915
   Introduction .................................. 1915
   Required Files .............................. 1916
   SPR Process Overview ....................... 1916
       Design Tips ............................. 1918
   Core Generation and Pad Routing .................... 1919
   Padframe Generation and Pad Routing ............. 1919
       SPR Port Annotation ...................... 1920
       Generating a Padframe from a Netlist with Pad Cells 1921
       Generating a Padframe Without a Netlist or Without Pad Cells 1921
   Global Input Signal Routing (Clock Routing) .... 1921
   SPR Setup .................................. 1923
       Mapping Table ............................ 1924
       Initializing Setup ....................... 1926
   SPR Core Setup ............................ 1926
       SPR Core Setup—General .................. 1927
       SPR Core Setup—Layers .................. 1928
           Over-the-Cell Routing ................. 1929
       SPR Core Setup—Design Rules ............... 1930
       SPR Core Setup—Placement ................ 1932
           Assigning Net Criticality .............. 1932
           Clustering Standard Cells .......... 1933
       SPR Core Setup—Global Signals .......... 1934
       SPR Core Setup—Power .................... 1935
       SPR Core Setup—I/O Signals .............. 1936
   SPR Padframe Setup ....................... 1937
       SPR Padframe Setup—General .............. 1937
       SPR Padframe Setup—Layout .............. 1938
           Adding Pads .......................... 1939
           Pad Naming and Ordering .......... 1939
L-Edit 16 User Guide—Contents

(Continued)

Mirroring ................................................................. 1941
SPR Pad Route Setup ................................................. 1941
  SPR Pad Route Setup—General ............................. 1943
  SPR Pad Route Setup—Layers ............................... 1943
  SPR Pad Route Setup—Design Rules ......................... 1944
  SPR Pad Route Setup—Core Signals ......................... 1946
  SPR Pad Route Setup—Padframe Signals ..................... 1947
Standard Cell Place and Route .................................. 1948
  Indent Middle Rows ........................................... 1950
  Placement Optimization ...................................... 1951
  Optimization Factor .......................................... 1951
Output Options ....................................................... 1952
  Label Nodes .................................................... 1952
  Nodal Capacitance Files (CAP) ............................. 1953
    Two-Layer Example ......................................... 1954
  Standard Delay Format Files (SDF) ......................... 1955
    Pin-to-Pin Delay Calculation ............................. 1957
SDF Driver Properties .............................................. 1958
  Import .LIB Timing Data ...................................... 1958
  Edit Pin Characteristics .................................... 1959
References ............................................................. 1959

  Standard Cell Library ........................................ 1960
  Standard Cells ................................................ 1960
    Abutment Ports ............................................. 1960
    Power Ports ................................................. 1960
    Signal Ports ............................................... 1961
    Row Crosser Ports ......................................... 1962
  Special Standard Cells ....................................... 1963
    Pad Cells ................................................... 1964
      Abutment Ports ........................................... 1964
      Connection Ports Between Pad Cells .................... 1964
      Signals from Pad to Layout Core ......................... 1965
      Power Supply Pads ....................................... 1965
      Corner Pad Cells ......................................... 1965
      Pad Cells Without Bond Pads ............................ 1966
      Pad Orientations ......................................... 1966
      Mirror Ports .............................................. 1966
  Designing Cells for Global Signal Routing .................. 1967
    Global Signal Port Definitions ............................ 1967
    Buffer Cell Input Ports .................................... 1968

A4  Place and Route File Formats ................................. 1969
  TPR Files ...................................................... 1969
    Syntax ....................................................... 1969
    Interpretation .............................................. 1970
  EDIF Files ..................................................... 1970
    Syntax ....................................................... 1971
    Interpretation: Pads ....................................... 1972
    Interpretation: I/O Signals ................................ 1972
Interpretation: Criticality .................................................. 1973
Additional Notes ......................................................... 1973
References ................................................................. 1974

SDF Files ........................................................................ 1974
Pin-to-Pin Delay Syntax .................................................. 1974
Interpretation ................................................................. 1974

CAP Files ....................................................................... 1975
Syntax .......................................................................... 1975
Interpretation ................................................................. 1975

B1 Netlist Extraction ....................................................... 1977
Legacy Layer Generation ............................................... 1977
Legacy Layer Generation ............................................... 1977
  Working with 45° Objects ............................................. 1978
  Writing Out Node Names ............................................. 1979
  Adding User Parameters to Extracted Devices ............... 1980

Legacy Extract ............................................................. 1982
  Setup Extract Standard Rule Set–General ..................... 1982
  Setup Extract Standard Rule Set–Output ...................... 1984
  Setup Extract Standard Rule Set–Subcircuit ................. 1986
  Activating Subcircuit Recognition .............................. 1987
  Using Subcircuit Definitions for Hierarchical Cell Recognition ............................................. 1988
  Designing Subcircuit Cells ........................................... 1988
  Subcircuit Recognition Polygons ................................. 1988
  Subcircuit Connection Ports ....................................... 1989
  Crossing Over a Subcircuit Instance ............................ 1992
  SPICE OUTPUT Properties ........................................... 1993
  Property Tokens .......................................................... 1993
  Application Example .................................................. 1994

Index ........................................................................... 1995

Credits ......................................................................... 2012
Database Options in L-Edit

Tanner EDA supports OpenAccess databases, an open standard data API and reference database for IC design access and supporting software. The OpenAccess database allows users to construct flows incorporating design tools from multiple providers, and to create collaborative designs with controls for input from multiple users.

L-Edit can use the Tanner tdb or an OpenAccess database. L-Edit looks and operates similarly regardless of the database you use.

OpenAccess is the preferred choice for users wanting multi-user support or the ability to use foundry process design kits in OpenAccess format. TDB is the preferred choice for projects with a single user and no needs for OpenAccess compatibility. OpenAccess databases have the ability to save data on a per cell basis, and thus offer fast save ability on large databases. TDB is a more compact database, and is faster for full database read and write operations.

“Design” versus “Library”

Designs and libraries in L-Edit are both collections of cells. The term “design” usually refers to the final product that is fabricated and “library” usually refers to component cells that are reused in many different designs. However, there is no distinction between the two in the way L-Edit operates. Designs or libraries can both instance cells from other designs or libraries.

OpenAccess Database Structure

In an OpenAccess database, each cell in a library is stored in a separate folder. Each view of a cell, such as the schematic view, layout view or symbol view, is stored in a separate sub-folder within the cell directory. Typically the filename in the view folder will indicate the view type, as in “layout.oa” or “symbol.oa,” as shown in the illustration below.
OpenAccess Cell Libraries

An OpenAccess design can contain cell libraries, and any library may be a cell library only, a technology library only, or both a cell and technology library.

The **lib.defs** file lists the name and path of all libraries used in a design, including the design library itself (see “lib.defs - the Library Path File in OpenAccess” on page 46). The **lib.defs** file may be edited directly in a text editor, or libraries may be added or removed in the **Setup > Design > Libraries** dialog. You can protect cells in a library against deletion or modification by setting operating system read-only or security settings on the files in a cell’s folder.

Alternatively, you can use the “Locked” flag in the **Library Information** dialog (accessed with **File > Info**) to lock all cells in a library, or the “Locked” flag in the Cell Information dialog to lock individual cells in a library.

Note that the library and cell lock flags will be modifiable by anyone editing the library, whereas disk security settings may be set on a per-user basis by a CAD manager.

Cell Reservations

When you start to edit a cell, you will be prompted to “reserve” that cell to prevent changes by other users. When you are done making changes, you save your changes to the database and release the reservation for the cell.
If you attempt to obtain a reservation on a cell that is already reserved, L-Edit can tell you who has the reservation. To obtain the latest updates made by other users, there are several update cell commands, and the Database toolbar provides shortcut buttons for reserving, saving and updating cells.

**TDB Database Structure**

Tanner Database (TDB) is a proprietary, machine-readable format optimized for the Tanner Tools environment. TDB files are saved with the `.tdb` filename extension.

Along with the design itself, a TDB file contains setup information including layer rendering information, CIF and GDSII setup information, design rules, and L-Edit configuration settings.

When a file is saved, L-Edit automatically backs up previously-saved versions of the file with a `.tdo` extension.

**TDB Cell Libraries**

A TDB design may contain cell libraries. Libraries can be added or removed with the `Setup > Design > Libraries` dialog. Only cells in the toplevel TDB design may be edited. Cells in TDB libraries may be instanced and viewed but not edited; they must be opened separately as a toplevel designs to be editable.

You can lock an entire library with the “Locked” flag in the `Library Information` dialog (accessed with `File > Info`) to lock all cells in a library, or you can lock individual cells in a TDB file with the “Locked” flag in the `Cell Information` dialog.

Note, however, that locked flags themselves may be changed by anyone editing the tdb file.

**Launching L-Edit**

To launch L-Edit, click the Start button on the Windows taskbar and navigate to L-Edit in the Tanner EDA directory. You can also double-click the L-Edit icon on your desktop, which looks like this:

![L-Edit Icon](image)

You also can launch L-Edit with a specific file:

- a `.tdb` file (Tanner database)
- a `lib.defs` file (OpenAccess)
- a `layout.oa` file from a view folder (OpenAccess)

Double-click on the file in Windows Explorer, use `File > Open`, drag-and-drop, or use the `most recently used` file list to find and launch either of these file types.

**Opening an OpenAccess Database**

An OpenAccess design can use any number of libraries. If a design consists of multiple libraries, L-Edit requires that one of the libraries be designated the top-level library. Since each library can have its own OpenAccess technology, the top-level library determines the root from which the OpenAccess technology is read. (Note that the top-level library may reference or attach to technology from other libraries.)
The list of and paths to the design and libraries are stored in a list file called lib.defs. (See “lib.defs - the Library Path File in OpenAccess,” below.)

When you open a design using a lib.defs file, L-Edit will prompt you to identify which of the libraries is the top-level. Libraries in the Confirm Top-level Library dialog are listed in the order in which they appear in the lib.defs file.

When you open a design using a layout.oa file, the library containing that file becomes the top-level library by default.

In a Tanner database, by default the design is the top-level library.

**lib.defs - the Library Path File in OpenAccess**

In OpenAccess, a library path file “lib.defs” specifies where to find the libraries that are used in a design. The library path file simply lists the libraries that should be loaded and made available for use when any particular design is opened. The library path file supports environment variables.

The format of a library path file is: DEFINE LibraryName LibraryPath, specifying the name of the library and the path to the library. For example:

```
DEFINE reference18nm ./basePDK/reference18nm
DEFINE RefCoreCells ./CoreLib/CoreCells
#DEFINE ViaLib $SCN_ROOT/db/ViaLib
```

Note that spaces may not appear as part of the Library Path—they are not supported.

The lib.defs file is very similar to the Cadence cds.lib file. In fact, if no lib.defs file is found you can browse to a cds.lib file instead. However, there may be some commands in a cds.lib file that are not supported in L-Edit.

When you open an OpenAccess design by opening a layout.oa file, L-Edit searches for a library path file in the following order:

1. Search up the disk path starting from the library containing the cell that was opened for the lib.defs file.
2. Search in the tanner application settings folder C:\Documents and Settings\<username>\Application Data\Tanner EDA for the lib.defs file
3. Use the path saved in the database, if there is one.

If no lib.defs file is found, L-Edit will prompt you to browse for a library path file. In this case, you can use any filename because when the design is saved, the path and filename you designate is also saved. If any of the paths defined in the lib.defs file do not exist, none of the libraries in the file will be loaded.

If a design uses a library but that library is not listed in the library path file, cells that are instanced from that library are designated unresolved references, and will appear as an outline crossed on the diagonal.

**Auto Loading TCL Scripts in an OpenAccess Database**

OpenAccess databases, in particular databases containing iPDK PyCells, may contain callbacks written in TCL. L-Edit provides a mechanism to automatically load tcl scripts when an OpenAccess databases is opened. A TCL file named autoload.tanner located in an OpenAccess library folder is automatically loaded when the library is opened. If autoload.tanner is not present, then a file named autoload.file is loaded.
A common use of autoLoad.tanner or autoLoad.file is to initialize the oaTcl subsystem, using the `package require oa` command. This is necessary for non-IPL compliant callbacks, which access OpenAccess data directly.

**File Contents in an OpenAccess Database**

When you set a technology reference for an Open Access design, it is useful to know where data is stored in an OpenAccess database.

- **tech.db** Stores the lists of Layers, Purposes, Layer Purpose Pairs, Valid Vias, and the collection of Via Templates (“via defs” in OpenAccess terminology).
- **data.dm** Library data such as author, organization, version, L-Edit “lock” status, and modification date are stored in the data.dm. data.dm contains references to other files that are under the data management control of OpenAccess, but are not native OpenAccess files. For example, the tech.tdb file is one such file that is registered with data.dm so that it can be locked, versioned, etc.
- **tech.tdb** Stores technology data that is not native to OpenAccess. It stores layer attributes such as rendering patterns, derivations and electrical properties. tech.tdb also stores Interactive DRC, Node Highlighting data, and Standard DRC rules.
- **lib.defs** Stores the list of library names and paths in an OpenAccess project.

**Using a Command-Line Argument to Launch a TDB File**

To launch L-Edit with a command-line argument, click the **Start** button on the Windows taskbar and select **Run**. Use the **Browse** button and navigate to the directory that contains `ledit.exe`. If a TDB file is not specified in the , L-Edit starts with an empty layout file similar to `ledit.tdb`.

Options can also be included in a program shortcut. L-Edit uses the following command-line arguments:

**Arguments** | **Description**
---|---
file1.tdb, file2.tdb, ... | The names of the TDB files to open. TDB files specified on the open with the number of layout windows they had when last saved.
-d | Prevents L-Edit from changing the current directory. Without this flag, L-Edit sets the current directory to that of the last TDB file that was opened in L-Edit.
-d `<dir name>` | Changes the current directory to the one specified.
-f | Instructs L-Edit to ignore workgroup and user configuration files. For more information, see “Workgroup and User Configuration Files” on page 61.
-n | Hides the splash screen. The L-Edit splash screen will not be displayed during product startup.
Establishing Technology Information

Every L-Edit design file needs basic information such as layer definitions, technology and fabrication settings, specific data for modules such as SPR or vias, etc. Collectively, this information is referred to as the technology information. Also, all geometry in an OpenAccess database is assigned and requires a layer number and a purpose number.

Technology information for a design may be saved locally within the library or outside of the design database.

When technology information is saved locally, whether you initially enter it manually or import it from another file, it is saved with the design file and can only be updated from within that file.

When saved outside the design, L-Edit retrieves the technology information each time a file is opened, using a technology reference.

The technology setup for a design can be a combination of referenced data and imported data. For example, you might reference an attached OpenAccess technology library and also import GDS.

OpenAccess Technology Libraries

An OpenAccess database may obtain technology information from technology libraries. Technology libraries may be linked either by Reference or by Attach. Referenced libraries are a list of libraries from which L-Edit reads technology information, in the order in which they are listed. Each time you load a design, L-Edit reads from the list of referenced libraries as specified.

If there is a conflict between two of the referenced libraries – for example, if a layer is defined differently – L-Edit will use the first value it reads.
When reading referenced libraries, L-Edit follows one full chain of hierarchy before proceeding laterally to the next chain of hierarchy. For example, in the diagram below, L-Edit will read the library hierarchy in the order A, B, C, D, E, F.

A design can reference many libraries but can only attach one library. Referenced or attached libraries may themselves reference or attach other libraries, so in either case there can be a tree of libraries.

A design with a referenced library has a top level tech.db, whereas a design with an attached library does not.

In both a referenced and an attached technology library, the lists of layers, purposes, layer-purpose pairs, and the collection of via templates is obtained by merging the tree of tech.db files, with precedence given to data higher in the tree.

However, the list of valid vias is read from the first library in the tree in which the list is found. No merging of valid via lists is performed. If the list of valid vias is modified, then the new list will be written to the toplevel tech.db in the case of a referenced technology library, and to the attached library in the case of an attached technology library. (Please see “Valid Vias” on page 396 for more information.)

In a design with referenced technology libraries, additions to these lists or collections by the user will be written to the toplevel tech.db, whereas in a design with an attached technology library there is no toplevel tech.db, so additions are written to the attached library’s tech.db. Deletions or modifications to items in these lists or collections is made from the library in which the item is defined.

**Protecting Your Data in Attached and Referenced Libraries**

You can protect data in the technology library from additions or deletions by making the tech.db in the referenced or attached technology libraries read-only, or by using disk security settings. In a referenced scenario with read-only tech.db in the tech libraries, the user will be able to add layer-purpose pairs, via templates, etc., because these will be saved to the toplevel.

In an attached scenario with read-only tech.db in the tech libraries, the user will not be able to add to lists or collections such as the layer-purpose pairs, as there is no toplevel tech.db.

Note that in either a referenced or attached technology case, the user may add or delete items from these lists or collections during an editing session—the read-only setting only protects what is written to the database.

Attributes of a layer-purpose pairs, such as rendering patterns, derivations, and electrical properties, are obtained by reading the tree of tech.tdb files until the layer-purpose pair data is found. The state of layer attributes are saved in the toplevel design tech.tdb whenever technology is saved, and will persist after save and reload operations if the toplevel tech.tdb is writeable.
Write-protecting the library will not prevent attributes of a layer from being modified. If design needs to be protected against modifications of layer attributes, then the toplevel tech.tdb should be made read-only.

Standard DRC, interactive DRC, and node highlighting setup data are stored in tech.tdb files. In both referenced and attached technology libraries, standard DRC, interactive DRC, and node highlighting data are obtained by first reading the toplevel tech.tdb file. The tree of tech.tdb files is then read and if data for any one of these tools is present, then the setup is replaced by the data in the technology library. Once a replacement from a library is performed, no further reading of the tree is done for that tool.

Standard DRC, interactive DRC, and node highlighting rules are saved to the toplevel tech.tdb when technology is saved. If no data is present in the tech library tree for standard DRC rules, interactive DRC, or node highlighting, then changes made in the toplevel design to these technology items are saved and will persist when the toplevel design is reloaded.

If data is present in the tech library for standard DRC Rules, interactive DRC, or node highlighting, then changes made in the toplevel design to these technology items will be replaced by the data in the library when the toplevel design is reloaded.

You can protect setup data in libraries by making the tech.tdb file in libraries read-only, as changes the user makes during an editing session will be replaced the next time the database is loaded. If no data for a tool is present in the library, then user changes will be allowed, saved and reloaded from the toplevel tech.tdb.

**Setting the Technology Reference in an OpenAccess Database**

You use the Setup Technology Reference dialog, invoked from Setup > Technology Reference, to choose where and also how technology information is referenced.
Available technology libraries

Lists all libraries from the lib.defs file or the Setup > Design > Libraries dialog.

Referenced libraries

Lists the technology libraries that will be referenced for the active design, in the order in which they will be read. Note that if a library in this list references other technology setups, those files are read and additively updated.

Add

Invokes a browser so you can add a library to the Available technology libraries list.

Include/Exclude

Include adds a library from the “available” list to the “referenced” list, Exclude moves it back.

Move Library

The library at the top of the list has the highest priority when applying rules and constraints to a design. Use the up and down arrow icons to set the order in which libraries will be referenced by moving them up or down in the list.

Technology Reference Options for OpenAccess

For an OpenAccess database, the technology reference menu options are as follows:

None

All technology information is obtained locally from the design.

OpenAccess referenced libraries (see “OpenAccess Technology Libraries” on page 48.)

The OpenAccess referenced libraries option let you create a list of libraries from which L-Edit reads technology information, in the order in which they are listed.

OpenAccess attached library (see “OpenAccess Attached Libraries” on page 53.)

The OpenAccess attached library sets one single library that will be accessed for technology information.
Virtuoso

Allows a technology reference to a Virtuoso technology (.tf) and display (.drf) file. L-Edit reads and includes layer purpose pairs, rendering patterns, via definitions, and design data, provided such information is in the files, when a database is opened.

Technology in the local database is read first, then the technology in the Virtuoso file is read and overwrites any conflicting data. In either an OpenAccess or TDB database tech reference, any changes to technology will be written to the design database. When the Virtuoso technology reference is read on opening the database, changes will be overwritten by the contents of the Virtuoso file. Changes in technology are not saved to the Virtuoso reference file.

Laker

Allows a technology reference to a Laker technology (.tf) and display (.drf) file. L-Edit reads and includes layer purpose pairs, rendering patterns, via definitions, and design data, provided such information is in the files, when a database is opened.

Technology in the local database is read first, then the technology in the Laker file is read and any conflicting data is overwritten. In either an OpenAccess or TDB database tech reference, any changes to technology will be written to the design database. When the Laker technology reference is read on opening the database, changes will be overwritten by the contents of the Laker file. Changes in technology are not saved to the Laker reference file.

TDB

References a Tanner (.tdb) file for technology information.

A TDB tech reference is read when a database is opened. Technology in the local database is read first, then the technology in the TDB file is read, overwriting data in the local database.

Layers may be added or deleted during an editing session, but when the database is closed and reopened, those changes will be replaced by the reference setup. Similarly, changes to node highlighting, interactive DRC, and standard DRC can be made during an editing session, but when the database is reloaded these changes will be replaced by the reference setup.

Note: In the case of a TDB reference, node highlighting, interactive DRC, and standard DRC rules are replaced by the reference setup, regardless of whether the referenced TDB contains data for these items or not. In other words, empty node highlighting data in the reference TDB setup will overwrite new settings a user makes during an editing session, when the dataset is reloaded.
OpenAccess Attached Libraries

An attached library is the one library that will be accessed for technology information. With an attached library no other technology reference is allowed, not even a local top-level file. However, the attached library can contain references of its own.

Technology Reference Options for a Tanner Database

For a TDB database, the technology reference options are None, Virtuoso or Laker technology (.tf) and display (.drf) files or a Tanner database (.tdb) file. These options behave as described in “Setting the Technology Reference in an OpenAccess Database” on page 50.

Importing Technology into a Design

Use Setup > Import Technology to import and save technology data from an external source into the active design.
You can import technology into L-Edit from a Virtuoso or Laker technology (.tf) and display (.drf) file, a Tanner database (.tdb) file or a GDS layer map file.

The Virtuoso and Laker import options behave as described in “Setting the Technology Reference in an OpenAccess Database” on page 50. The TDB option opens the Import Technology dialog. The GDS option lets you choose a GDS map file that contains layer names, purpose names, GDS numbers and the GDS datatype.

**Importing Technology from a TDB File**

Setup > Import Technology lets you import setup information from a TDB file to the current file.

**Import Technology**

- **Technology**:
  - TDB

- **Layers**:
  - Layers
  - Replace
  - Merge

- **Design**:
  - Technology
  - Maintain physical size

- **Modules**:
  - DRC rules
  - Extract
  - Cross-Section

**From file**

Name of the TDB file whose setup is to be imported. Click **Browse** to navigate to an existing file.
Layers

Imports layer setup from the specified file.

- **Replace** deletes the layers in the destination file and replaces them with the layers from the source file.

- **Merge** adds the layers from the source file to the list of available layers in the destination file. Source file layers not present in the destination file are appended to the layer list in the destination file. If a layer in the source file has the same layer number as a layer in the destination file, the position it has in the destination file is maintained. For further information, see “Layer Merging During Imports” on page 59.

Additional layer-specific setup options include:

- CIF names
- GDS II numbers
- Wire settings

Technology

Options include:

- **Maintain physical size**—With this option, L-Edit checks all objects in all cells and unit-specific parameters entered in other dialogs to determine if the layout will be truncated when it is rescaled. L-Edit presents one warning for each cell and set of parameters if a truncation will occur. If you answer **Yes** to all the warnings, or if no truncation will occur, L-Edit rescales the design. If you answer **No** to any of the warnings, L-Edit cancels the rescaling operation.

- **Rescale**—L-Edit rescales the design by applying technology scaling parameters in the source file to objects in the destination file.

Draw

Transfers the parameters entered in *Setup Design–Drawing*.

Palette

Transfers the color parameters entered in *Design Setup*.

Show/Hides

Transfers the view settings for grid, origin, ports, and other objects.

Properties

If checked, replaces the System and other parameters set in *File > Info—Properties*.

Selection

Transfers the parameters entered in *Setup Design—Selection*. See “Setup Design–Selection” on page 85.

Grid

Transfers the parameters for the display grid and mouse snap grid. See “Setup Design–Grid” on page 82.

Object Snap

Transfers the setup parameters for object snapping. See “Setup Design–Object Snap” on page 88.

Interactive DRC

Transfers the setup parameters for interactive DRC. See “Setting Up Interactive DRC Rules” on page 510.

Node Highlighting

Transfers the setup parameters for interactive DRC. See “Node Highlighting Setup” on page 517.
Chapter 1: Library and Design Configuration

Importing Technology into a Design

**Importing a Setup from Virtuoso**

The **Setup > Import > Import Virtuoso® Setup** command simplifies transitions of designs from the Cadence Virtuoso® to the Tanner L-Edit design environment. Importing a Virtuoso setup always creates a new L-Edit file. The display and tech files are concatenated and read as one.

Imported elements from a Virtuoso technology file include palette colors, background color, grid colors, layers (including rendering information, GDS layer number and GDS datatype) and manufacturing grid. L-Edit elements that are missing from Virtuoso (such as user-defined rendering) are automatically generated.

**Display file**

Specifies rendering information (palette colors and stipple patterns.)

**Tech file**

Specifies technology information (layers, purposes, and bindings between these and display data.)

**Importing a Setup from Laker**

You can import Laker format file setups to the Tanner L-Edit design environment using **Setup > Import > Import Laker Setup**. You must have a file open in L-Edit to access the command, but importing a Laker setup creates a new L-Edit file. The command creates an L-Edit technology setup by reading the Laker display (.dsp) and technology (.tf) files. Imported elements include palette colors, background color, grid colors, and layer rendering information, GDS layer number and GDS datatype, guard rings and contact cells, and DRC rules.
L-Edit elements that are missing from Laker (such as user-defined rendering) are automatically generated with default settings. L-Edit will warn if it encounters an undefined command, but the import will continue.

**Importing Technology into a Design**

L-Edit provides a brief summary after a successful import.

**Importing a GDS Layer Map**

Use **Setup > Import Technology > GDS Layer Map** to import GDS numbers and datatypes for layer purpose pairs from the GDS layer map file you specify.
GDS Layer Map File Format

The layer map file sets the mapping from layer purpose pairs to GDS layer and datatype as follows:

<table>
<thead>
<tr>
<th>Layer-Name</th>
<th>Purpose-Name</th>
<th>GDS-Number</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poly</td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Active</td>
<td></td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Metal1</td>
<td>drawing</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Metal2</td>
<td>drawing</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

It is possible to map a single layer to multiple stream layers. For layers that do not have a purpose name, simply omit the purpose name.

Importing CDF

Import CDF can be used to import parameters into a T-Cell from a Cadence Design Systems CDF file. To do this, invoke Setup > Import Technology and choose CDF.

- **File**
  - Name of CDF file to import. A single CDF file may contain parameters for multiple libraries and cells, with headers that indicate the cell and library for the parameters that follow.

- **Directory**
  - Directory name containing CDF files. A directory may contain multiple CDF files, each one containing parameter data for a single or multiple cells.

- **Extension**
  - File extension of CDF files to read in the specified directory.

- **Also search subdirectories**
  - When this option is checked, L-Edit will also read CDF files in subdirectories below the specified directory.

- **Create cells missing in design**
  - When this option is checked, L-Edit will create a new cell if the cell in the CDF file does not exist in the database.
Layer Merging During Imports

When you merge layer setups, L-Edit adds source-file layers to the layer list in the destination file. If the source file has layers not present in the destination file, L-Edit appends them to the destination-file layer list. If the source file and destination file have a layer with the same layer number, the layer maintains its position in the destination file’s layer list.

For example, a source file contains layers A, B, and C (in that order), and a destination file contains layers B, D, and E (in that order). After replacing, the destination file will contain layers A, B, and C. After merging, the destination file will contain layers B, D, E, A, and C. (The destination file’s information on layer B is replaced with the source file’s information on layer B.)

Note: Importing a layer setup also transfers a layer’s lock status from the source file to the target file. For example, if you lock Metal1 in the source file, it will be locked in the target file. Conversely, if you lock Metal1 in the target file but unlock it in the source file, it will then be unlocked in the target file after you replace the layer setup.
2 Application and Design Setup

Setting the Number of Available Colors

You can display an L-Edit design file using a palette of 16, 32, 64, 128, or 256 colors. Each color is defined by two attributes, a unique identifying binary code and an RGB color definition, using the Setup > Colors command.

![Setup Colors dialog](image)

**Number of colors**
Select the number of colors that will be available for defining layer colors. Options are 16, 32, 64, 128, or 256 (True Color mode) colors.

**Sort colors by**
Select how colors will be sorted. This setting applies to both this dialog and the Setup Layers dialog. Options are:

- **Index**—sorts by index number, which is the binary value of a color.
- **Number of bits**—sorts by the number of bits used to define a color and then by index number if the number of bits set is equal.
- **Hue**—sorts by hue, then saturation, then luminosity, then index number.
- **Brightness**—sorts by luminosity in descending order, then hue, then saturation, then index number.

**Binary code for the color.**

**RGB values for the color, in this case R=85, G=170, B=170.**
Application Setup

Application-level settings control primary L-Edit interface options. They are divided into ten separate tabs—**General**, **Editing**, **Keyboard**, **Mouse**, **Warnings**, **UPI**, **Rendering**, **Selection**, **Text Editor** and **Text Style**.

Application Configuration Files

Application settings are saved in application configuration (.ini) files, which are ASCII files that can be edited and shared by multiple users.

To load settings from an existing file, enter the name of the file in the **Workgroup** or **User** field, or choose from available files using the **Browse** button next to the desired field. Click **Load** to read the settings into L-Edit.

Workgroup and User Configuration Files

L-Edit can load configuration information from either a Workgroup or a User file. **Workgroup** files are intended to be shared by multiple users; for example, they may contain key remapping sequences that will be used by many users. **User** files are intended to contain preferences specific to a particular individual.

When both a workgroup and user file are specified in **Setup Application**, settings from the user file take precedence over settings in the workgroup file.

To create a workgroup configuration file, first save the desired settings in a user configuration file. You can then copy the user configuration file with a new name to create a workgroup file.

Changes to the **Setup Application** dialog can only be saved to a user configuration file. An .ini file loaded as a workgroup file cannot be overwritten.
The default location of the `ledit.ini` settings file on Windows Vista is in the application data `AppData` folder for `Tanner EDA` or wherever your specific company has placed user-specific configuration data, for example:

```
C:\Users\<user>\AppData\<Roaming>\Tanner EDA
```

The default location of the `ledit.ini` file on Windows XP is also the `AppData` folder for `Tanner EDA` but under Documents and Settings:

```
C:\Documents and Settings\<user>\Application Data\Tanner EDA
```

**Editing Configuration Files**

Tanner INI files use the Windows INI file format, which can be edited with any text editor. To write out the user configuration file, press `Shift+Enter` or hold the `Shift` key while clicking `OK` in the `Setup Application` dialog.

**Contents of Configuration Files**

Some of the L-Edit parameters saved in an INI file are as follows:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Refer to Section:</th>
</tr>
</thead>
<tbody>
<tr>
<td>General options, including:</td>
<td></td>
</tr>
<tr>
<td>• Mouse settings</td>
<td>“Setup Application–Mouse” (page 68)</td>
</tr>
<tr>
<td>• Recently used file list and TDB Setup Path</td>
<td>“Setup Application–General” (page 63)</td>
</tr>
<tr>
<td>• Enable keyboard shortcuts in the Design Navigator</td>
<td>“The Design Navigator” (page 242)</td>
</tr>
<tr>
<td>• Show Verification Error Navigator after running DRC</td>
<td>“Optimizing Performance” (page 545)</td>
</tr>
<tr>
<td>• Toolbar settings</td>
<td>“Setup Application–General” (page 63)</td>
</tr>
<tr>
<td>Editing options, including:</td>
<td></td>
</tr>
<tr>
<td>• Autopan, rubberbanding, Paste to cursor</td>
<td>“Setup Application–General” (page 63)</td>
</tr>
<tr>
<td>• Instance rendering and caching</td>
<td>“Setup Application–Rendering” (page 72)</td>
</tr>
<tr>
<td>Warnings</td>
<td>“Setup Application–Warnings” (page 70)</td>
</tr>
<tr>
<td>CIF import and export options</td>
<td>“Importing CIF Files” (page 136) and “Exporting CIF Files” (page 148)</td>
</tr>
<tr>
<td>GDSII import and export options</td>
<td>“Importing GDS Files” (page 133) and “Exporting GDS Files” (page 144)</td>
</tr>
<tr>
<td>Keyboard remapping settings</td>
<td>“Setup Application–Keyboard” (page 67)</td>
</tr>
</tbody>
</table>
Setup Application—General

Use the **General** tab to customize toolbar display, language display, and other general application parameters.

- **Layer icon size**: Sets the default pixel size of icons on the Layer palettes. You can increase or decrease the icon size in two-pixel increments at any time by right-clicking on any layer name, in either Palette, and selecting **Options**.

- **Drawing mode**: Sets the default display of drawing tools on the Drawing toolbar. When **Orthogonal** or **45 Degrees** is chosen, only those tools fitting that description will be displayed. When **All Angle** is selected, all tools are displayed.

- **Use large buttons**: When checked, increases the size of all toolbar buttons by 50 percent.

- **Recently used file list**: Controls the number of recently used files displayed in the **File** menu.

- **Language used in menus**: Select the language for menus. Choosing Windows in this entry will put menus in the language based on Windows settings.

- **File Integrity**: If checked, L-Edit reads the tdb file immediately after writing. If it detects errors it reports them and reverses the write operation by renaming the .tdx (backup of backup) file to .tdo, and the .tdo (backup) file to .tdb. As noted, this can be a time consuming process.
### When opening a view of a cell

Controls if the active window is used, or a new window is opened when opening a cell. These options are also used when cells are opened when selecting objects in the Hierarchical Find list, and also when cross probing from S-Edit.

- **Reuse the active window** sets L-Edit to replace the contents of the active layout window when opening a new cell.
- **Open a new window** sets L-Edit to open a new layout window each time you open a new cell view.

**Note:** Holding the `Ctrl` key down while opening a cell will force it to open in a new window.

### Remove reservation after saving

These checkboxes control the removal of the reservation when saving an OpenAccess database. Both, one, or neither option may be checked. If neither option is checked, then the reservation on a cell is kept when a cell is saved.

- **Modified cells** — If the Modified cells option is checked, then the reservation on a cell is removed when saving the cell if the cell has been modified.
- **Unmodified cells** — If the Unmodified cells option is checked, then the reservation on a cell is removed when saving the cell if the cell has not been modified.

### TDB setup path

Predefined directories for TDB setup files. TDB files in these directories are listed in the **Copy TDB setup from file** field in the File > New and File > Import Mask Data dialogs.
Use the **Editing** tab to customize editing options, locator bar display, and functions triggered by the middle mouse button.

![Setup Application dialog box](image)

- **Paste to cursor**: When this option is checked, objects placed in the layout with `Edit > Paste` move with the pointer until any mouse button is clicked. They are then “dropped” into place at the location of the pointer (see “Paste to Cursor Feature” on page 329.) Before objects are dropped in their final position, they can be rotated or flipped using keyboard command shortcuts (see “Reorienting” on page 326.)

- **Auto-panning**: When this option is checked, L-Edit automatically pans the view when the pointer touches an edge of the window during a draw, move, or edit operation.

- **Active-push rubberbanding**: When this option is checked, it is unnecessary to hold down the mouse button during a drag operation. For example, when drawing a box, you can click and release the DRAW button at one corner of the box, move the pointer to the opposite corner of the box, then click the DRAW button again to complete the operation.

- **Show edit vector**: When this option is checked, L-Edit draws a rubberband line during a move or edit operation from the cursor start position to its current position during the click-and-drag operation.
**Increment port text on Duplicate**

When this option is checked, and the last character in the port name field is a numeral, L-Edit automatically increments the port label by one with each port that is placed.

**Instance stretching**

When this option is checked, instances and arrays will be stretched, where stretching increases the repeat count of the instance or array in the direction of the stretch.

The default state of this option is on, which sets the middle mouse button in “EDIT” mode when the cursor is on the corner or edge of an instance’s minimum bounding box. The instance will “MOVE” when the cursor is not on an edge or corner, or when this option is set to “OFF.”

- For instances of regular cells, stretching converts the instance into an array.
- For instances of T-Cell, stretching regenerates the instance with updated stretching parameters. (T-Cell parameters have an option for this purpose.)
- For arrays, stretching increases or decreases the number of rows and columns in the direction of the stretch by the whole number that best fits the extent of the edit.

**Locator bar display during editing**

These checkboxes set the type of coordinates the locator bar displays in the editing windows.

Pressing Shift+F will cycle through the different locator bar display types (also View > Display > Cycle Locator Bar Format.)

Pressing Q will toggle the locator bar display to relative coordinate mode with respect to the position of the cursor when Q was pressed. When you press Q again, the display goes back to the default mode.

- **Delta format**—displays coordinates in (delta-x, delta-y, delta) format relative to the start of the edit.
- **Polar format**—displays coordinates in polar format (delta, angle) relative to the start of the edit. When an edit operation is not being performed, the polar coordinate display is relative to the origin of the cell.
- **Absolute format**—displays the value of the x- and y- coordinates of the current position of the mouse.

**Ctrl+Shift+Middle Mouse action**

Sets the behavior of perpendicular edge movement when you use the middle mouse button together with the Ctrl and Shift buttons.

- **Preserve angles**—when you move an edge by holding the CTRL and SHIFT keys while doing a middle-button drag, L-Edit will preserve the angle between that edge and the adjacent edges.
- **Preserve edge length**—when you move an edge by holding the CTRL and SHIFT keys while doing a middle-button drag, L-Edit will preserve the length of that edge.
Use the **Keyboard** tab to customize keyboard shortcuts.

[Image of the Keyboard tab interface]

- **Category**: Command categories corresponding to L-Edit menu items. To select a category, highlight an item in the drop-down list.
- **Commands**: The set of commands for the selected category. Highlight a command from the list to select it. Additional commands may be viewed by moving the scrollbar up or down.
- **Description**: A description of the command highlighted in the **Commands** list.
- **Current keys**: The current shortcut key combination for the command highlighted in the **Commands** list. To delete a shortcut, highlight the shortcut and click **Remove**.
- **Press new shortcut key**: With the cursor in this field, press the desired shortcut on the keyboard, which will then be textually represented in the field. Click **Assign** to register the new shortcut.
- **Default**: Reassigns all shortcut key assignments in the current editor to their default settings.

Not that only user-defined keyboard assignments are saved in the user configuration file. To write all keyboard assignments to the user configuration file, press **Shift+Enter** or hold the **Shift** key while clicking **OK**
Setup Application–Mouse

Use the Mouse tab to customize mouse button assignments and mouse tooltip display options.

**Show under mouse pointer**

When checked, the mouse pointer displays a persistent tooltip showing the operation associated with each mouse button.

- **Text only** displays vertical text describing each button operation.
- **Pictures only** displays icons indicating each button operation.
- **Both text and pictures** displays tooltip text and icons with the mouse pointer.

When **Solid Background** is checked, L-Edit encloses the mouse tooltips in a box with solid fill.

**Note:** You can toggle mouse tooltips display at any time using **View > Display > Mouse Hints.**

**Button Assignments**

Determines the order in which mouse button assignments are displayed in tooltips and in the Mouse Button bar. You can select a **Right-handed** or **Left-handed** mouse, or choose **OS default** to use the default setting for your operating system.
**Hide middle button operations**

When selected, hides the middle button reference in the mouse tooltips and the Mouse Button Bar. Use this feature to show the functionality of a two-button mouse.

**Scroll wheel**

These controls set the default mouse wheel behavior to zoom or pan.

Pressing the Ctrl button while you use the mouse wheel toggles its function from one mode to the other. The zoom setting can be additionally controlled to make an upwards roll zoom in or zoom out.

- **Zoom in** and **Zoom Out** set the default behavior to zoom and additionally determine whether rolling the wheel upward zooms in or zooms out.

- **Pan up** sets the default mouse wheel function to a vertical pan (up and down.) When this option is selected, you can use the Shift button together with the mouse wheel to toggle to a pan horizontal pan (left and right.)

When **Ignore wheel while drawing and editing** is checked, the mouse wheel is disabled during drawing and editing operations. When it is unchecked, the mouse wheel controls described above are all functional in the drawing or editing modes.

**Full-screen cursor**

When this option is selected, the cursor is displayed as vertical and horizontal lines extending across the full extent of the layout window.
Setup Application–Warnings

Use the **Warnings** tab to enable and disable optional warnings and explanations that you may encounter while editing a design file.
Choosing the **UPI** tab in the **Setup Application** dialog allows you to set a path to the header files that L-Edit uses when running an interpreted macro. This dialog is also used to set a path to the log file where UPI writes macro errors.

---

**Location of the interpreter header files**

The complete path of the directory containing the L-Edit interpreter header files. Clicking **Browse** next to this field calls a standard Windows directory browser.

*Note:* The path name in this field is limited to a maximum of 75 characters. Therefore, you should not keep interpreter header files in highly nested subdirectories.

**Update display**

When checked, L-Edit updates the display while UPI code is executing. When unchecked (default), L-Edit does not update the display during the execution of a macro or T-Cell generator.

**Show warning dialog boxes**

When checked (default), L-Edit displays warning dialog boxes in the user interface. When unchecked, L-Edit runs in *quiet mode*, in which warning dialog boxes are not displayed.

*Note:* Batch processing must be run in quiet mode.
Setup Application–Rendering

Use the **Rendering** tab to establish basic display behavior.

### Hide instance insides if less than

Defines the minimum size (in pixels) that an instance must be to be completely drawn on the screen. If the instance width is smaller than the **Horizontal** parameter or the instance height is smaller than the **Vertical** parameter, it is drawn in outline mode—its insides are not shown.

Suppressing the display of instance insides can enhance screen redraw times and clarify the layout if the screen is zoomed out to a relatively small magnification.

### Cache instances smaller than

When checked, defines the maximum size (in pixels) that an instance may be to be cached. Cached instances are rendered once and then copied for each instance reference. They can be subsequently redrawn at much higher speeds than non-cached instances but may display alignment artifacts at the instance boundaries.

Very large cached instances may exhaust the available memory. The default maximum size is 700 pixels.

### Hide objects smaller than

When checked, defines the minimum size (in pixels) that an object must be to be drawn on the screen.
<table>
<thead>
<tr>
<th>Redraw</th>
<th>Choose <strong>All windows</strong> to redraw, for example, instances of a cell in other windows when the cell itself is modified, or <strong>Active window</strong> to redraw just what is displayed in the currently active window. L-Edit will redraw other windows as soon as they are activated. Redrawing the active window only can substantially improve rendering speed.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fill objects when editing/drawing</strong></td>
<td>When unchecked, renders objects in outline mode during drawing or editing operations.</td>
</tr>
<tr>
<td><strong>Rendering can be interrupted</strong></td>
<td>When checked, rendering can be interrupted with any mouse click or key stoke so that a full redraw does not have to be completed between each operation.</td>
</tr>
<tr>
<td><strong>Show design while rendering</strong></td>
<td>When checked, periodically updates the design display during rendering. The <strong>First time after</strong> field specifies the elapsed time from the start of a rendering operation to the first display update. L-Edit then periodically updates the display at the interval specified in <strong>Then every</strong> until rendering is complete. Both times are defined in msec. When <strong>Show design while rendering</strong> is not checked, L-Edit waits to complete a rendering operation before updating the display. This setting allows for faster rendering, but with less immediate feedback during screen refresh operations.</td>
</tr>
<tr>
<td><strong>Advanced performance settings</strong></td>
<td>Opens <strong>Advanced Performance Settings</strong>, which allows you to set and test system performance options that affect L-Edit rendering speed. (See “Advanced Performance Settings,” below.)</td>
</tr>
<tr>
<td><strong>Instance Name Text Height</strong></td>
<td>These fields set the font size of cell names so they will remain visible regardless of zoom level.</td>
</tr>
<tr>
<td><strong>Temporary rulers text height</strong></td>
<td>Enter a size in pixels for temporary ruler text.</td>
</tr>
</tbody>
</table>

The **Maximum** value sets the largest font height of the instance name, in pixels. (If this value results in an instance name that will not fit in the instance MBB L-Edit will shrink it to fit.)

The **Minimum** value sets the size at which instance name text display will be suppressed, in pixels.
### Advanced Performance Settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use CPU for color mixing</strong></td>
<td>Affects rendering performance by allowing your CPU or video card to handle the process of color mixing. L-Edit mixes layer colors to produce the proper color display where objects overlap. When relatively few objects are drawn on a relatively large number of layers, performance can degrade. Unless you have very powerful video card, this option should be checked to perform color mixing on the CPU.</td>
</tr>
<tr>
<td><strong>Use MMX technology</strong></td>
<td>If the CPU is used for color mixing, MMX technology generally provides optimal performance. However, occasionally an unusual system configuration produces better performance with this option disabled.</td>
</tr>
<tr>
<td><strong>Use PatBlt function for patterns</strong></td>
<td>L-Edit uses a fast Win32 function called PatBlt when rendering layers with objects having patterned fill or outlines. However, video driver manufacturers may implement this function in a non-optimal or unpredictable manner. If you find that patterns are not rendered correctly when this option is checked, try unchecking it. Note that for Windows 95/98/ME it is recommended that this option be disabled.</td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>Restore the default settings, which are best for most system configurations.</td>
</tr>
<tr>
<td><strong>Test</strong></td>
<td>Performs a rendering test by measuring how long it takes to redraw the top layout window ten times. Use this test to analyze different performance settings.</td>
</tr>
</tbody>
</table>
Setup Application—Selection

Use the **Selection** tab to specify selection modes and general selection parameters.

Note that if you simply click the mouse repeatedly without moving it, L-Edit will select nearby objects in a proscribed order, as described in “Cycle Selection” on page 312.

When this option is enabled, locked instances and objects on locked layers can be selected and their properties can be inspected.

This setting can be modified for an individual instance from the **Edit Object** dialog.
**Minimum zoom/selection box size**

Specifies the minimum area, in pixels, of a zoom or selection box. The zoom/selection box is a construct L-Edit uses in zooming to a specified view and selecting objects.

A *zoom box* defines the boundaries of the view during a zoom operation. When you choose View > Zoom and draw a box, for example, L-Edit zooms to a view corresponding to that box.

A *selection box* specifies an area within which L-Edit selects objects. When you drag a selection box around an object, selects all or part of that object, depending on your Edge selection mode setting.

If your mouse is not perfectly stable, you can use the option Minimum zoom/selection box size to specify the minimum size of this box. Use a relatively small value, such as 2 or 3 pixels, to prevent L-Edit from misinterpreting small, accidental mouse movements as a zoom or selection box.

**Object selection**

Use these options to determine whether an object must be fully enclosed by a selection box in order to be selected, or whether you can select an object when it is only partly enclosed by the selection box.

**Edge selection**

Use these options to determine whether the edge of an object must be fully enclosed by a selection box in order to be selected, or whether you can select an edge when it is only partly enclosed by the selection box.

- **Select edges only when fully enclosed by selection box**—with this option, L-Edit will only select object edges only if they are completely contained within the selection box. Any previously selected objects will be deselected.

- **Select edges when partly enclosed by selection box**—with this option, L-Edit will select all edges of an object completely or partly contained within the selection box.

**Enable implicit selections**

When implicit selection is enabled, a click-and-drag of the middle mouse button while near or over an object or object edge will implicitly select that object. See “Selection and Deselection Ranges” on page 86 for how to set the range of implicit selection.

When this option is not checked, you must select objects explicitly.

**Highlight implicit selections**

Implicit selections are highlighted with dashed outline as the mouse is positioned over an object or an edge. This gives an indication which object or edge an implicit edit will act upon. A useful feature when this option is on is that cell and instance names will be displayed when the mouse moves over an instance.

**Automatic Viewport change**

Use these options to control panning and zooming of the target cell when probing to a cell from S-Edit, Finding objects in Hierarchical Find, or browsing errors from the Verification Error Navigator.
Setup Application—Text Editor

Use the **Text Editor** tab to control whether and how files are saved when a file has been modified outside of the L-Edit text editor.

When the L-Edit text editor loads a file it does not lock it. It checks the stored version of a file for modifications when files are saved, first changed, and when the text window or application becomes active or is closed. If a stored file has not changed, nothing will happen.

**Auto-Load**

When enabled, L-Edit automatically updates the text files that have been modified outside of the text editor when those files have not been modified within the text editor.

**Modified text files**

If documents that will be used during DRC, extract, LVS, simulation or UPI have been modified before any of these operations are started:

- **Save all changes**—silently saves only those files directly associated with the operation. Does not save any files that are indirectly associated with the operation such as include files.
- **Prompt to save changes**—opens a dialog indicating which files have been modified, with the option to save them.
- **Don't save changes**—modified files will not be saved and the operation will use the stored version of those files.
Files Modified Outside the Text Editor

When **Auto-Load** is disabled, L-Edit will open a checklist of all the files open in the text editor that have been modified elsewhere. You will have the option to reload modified files (checked) or not. Files that have also been modified in the text editor will be highlighted. Similarly, when **Prompt to save changes** is selected, L-Edit will open a checklist of the modified files associated with the operation you are running.

![Reload Modified Text Files dialog box](image)
Setup Application–Text Style

Use the **Text Style** tab to define the types of text (keyword groups) that will be highlighted in the text editor, and their appearance.

Each file type has a set of predefined keyword groups that cannot be edited or deleted. Use this tab to view those settings, and to add or remove your own keyword groups with customized characteristics.

- **File Type**: A drop down list of the file types for which keywords are or can be defined.
- **Font**: Allows you to set the typeface (**Face Name**) and point **Size** in which a given keyword group will appear.
- **Paragraph**: Allows you to set the increment, in spaces, of the **Tab Size** used by the text editor.
- **Groups**: Displays the keyword groups defined for a given file type. Use **Add** to enter the name of a new keyword group. Use **Edit** to enter the keywords belonging to a group. Use **Remove** to delete a keyword group.
- **Colors**: Use **Foreground** and **Background** to set the respective colors for a keyword group.
Adding Keywords to a Group

Edit keyword groups opens the Keywords dialog, which allows you to enter keywords and to specify whether the case is evaluated (Case sensitive keywords checkbox enabled) when highlighting is applied.

Design Setup

To modify design-level settings in L-Edit, choose Setup > Design. This command opens the Design Setup dialog, which has tabs that allow you to manipulate design parameters in eight categories:

Internal Units, Display Units, and Technology Units

L-Edit uses display units to report object dimensions and coordinates. The program also uses display units to set the display grids and mouse snap grid. You can choose to display units of microns, mils, millimeters, centimeters, inches, or a custom unit. (Custom units can be defined in the Setup > Grid—Technology dialog.)

Note that the choice of display units does not affect the scaling of your design.

For its own computation, L-Edit uses internal units (30-bit signed integers). Before beginning your design, it is important that you define the relation between L-Edit’s internal units and physical, or technology, units, as this will determine the extent of the layout area and the smallest object that can be drawn. This relation is also critical when you replace your design setup or export a design to CIF or GDSII format. Defining this relationship sets the scale of the design file.

The L-Edit layout area extends from -536,870,912 to +536,870,912 internal units in both the x- (horizontal) and y- (vertical) directions. Thus, if 1 internal unit = 0.001 micron, the largest possible design is 1,073,741 microns (almost 42.3 inches) on a side. Similarly, the smallest dimension L-Edit can define is 1 internal unit. If 1 internal unit = 0.001 micron, the smallest possible feature size would be 0.001 micron. (In practical terms, of course, 0.001 micron is an unrealistically small feature size.)

In practice, you might also wish to adjust other settings based on your minimum feature size. If you use the display grid as a visual aid while drawing, you may wish to adjust its spacing. To achieve adequate
resolution, you may wish to adjust the spacing of the mouse snap grid or turn it off altogether. You can enter new values for these settings on the **Setup Design—Grid** tab. If you change the display units, L-Edit will automatically convert these numbers to the new unit system.

See “Setup Design—Grid” on page 82 for further information on setting the display grid and the mouse snap grid.

**Setup Design—Technology**

To specify technology parameters, choose **Setup > Design**. The **Setup Design** dialog with the **Technology** tab displayed appears:

![Setup Design Generic_250nm_TechSetup](image)

- **Technology name**
  - Used to determine whether two design files are compatible. If you attempt to copy a cell from a file with a technology name different from that of the current cell, L-Edit presents a warning.

- **Display units**
  - Specifies the units that L-Edit uses in displaying distance and area values, as well as the units in which physical distances are specified in other user dialogs. For example, you may wish to define a technology unit that corresponds to a manufacturing specification, such as a fraction of a micron. You can still choose display units of microns, so that all distances will be displayed in a familiar unit system.

  Changing the display units does not change the scale of your design. You can use any of the predefined units (**Microns**, **Mils**, **Millimeters**, **Centimeters**, or **Inches**), or a custom unit (if one is defined).
Technologies defined by a specific unit of measurement. Select one of the predefined units (Microns, Mils, Millimeters, Centimeters, or Inches), or a custom unit (Other). If you choose a custom unit, you must also specify its equivalent in microns and in internal units (for CIF/GDS II output, design rule checking, and other purposes) under Technology Setup. See “CIF File Formatting” on page 153 or “GDSII File Formatting” on page 158 for more information.

Database Resolution
Defines the relationship between internal units and technology (physical) units.

Note: Changing the database resolution will rescale your design. This operation cannot be undone.

Technology to micron mapping
For custom units, defines the relationship between microns and the custom unit. For example, you might want to work in technology units equal to a fraction of a micron, such as 0.18. Under Technology Units, click Other. You can type a name for the new units, or accept the default name of Lambda. Then, under Technology to micron mapping, define 1 Lambda = 18/100 Microns.

Setup Design—Grid
To aid the viewing, drawing, and editing of objects, L-Edit provides three independent grids—the displayed grids, the mouse snap grid, and the manufacturing grid—each of which divides the layout area into equal squares whose corners are gridpoints.

- The display grid provides a set of convenient locating points. L-Edit can display both a major grid and a minor grid.
- The mouse snap grid determines the pointer’s freedom of movement.
- The manufacturing grid corresponds to the resolution at which the manufacturer can produce circuit elements.

Grid parameters are specified using Setup > Design—Grid. Parameters that represent physical distances are always specified in the current display units. You can set display units on the Setup Design—Technology tab. When you change display units, L-Edit automatically converts the grid settings to the new display units.

See “Setting and Displaying the Grid Layer” on page 84 also.
Setup > Design—Grid

**Major displayed grid**

The absolute spacing of the major grid display. The value entered in this field is the distance, in display units, between major grid points.

**Suppress major grid if less than**

The apparent spacing of the displayed grid varies with the magnification of the Layout Area. If the number of screen pixels between major grid points is less than the value entered in this field, then the major grid is hidden.

**Minor displayed grid**

The absolute spacing of the minor grid display. The value entered in this field is the distance, in display units, between minor grid points.

**Suppress minor grid if less than**

The number of screen pixels between minor grid points below which the minor grid is hidden.

**Cursor type**

- **Snapping** — Causes the cursor to snap to the gridpoints specified in **Mouse snap grid**.
- **Smooth** — Allows the cursor to move unconstrained. Points picked during drawing and editing operations are still snapped to the mouse snap grid when the cursor type is Smooth.
Setting and Displaying the Grid Layer

L-Edit renders both major and minor grids on a Grid layer. Use Setup > Special Layers to specify the which layer will be the grid layer. Use Setup > Layers—Rendering to set the grid color by setting the rendering parameters of the grid layer.

The major and minor grids take the colors specified for Object Outline and Object Fill, respectively, on the grid layer. If there is no outline, both major and minor grids are displayed in the Object Fill color. Similarly, if there is no fill, both grids are displayed in the Object Outline color. If neither fill nor outline exists on the grid layer, then no grids are rendered.

You can toggle grid display on and off from the layout with a check mark in View > Display > Major Grid and View > Display > Minor Grid.
You can modify object selection parameters in the Setup Design—Selection dialog.

**Selection range**
A positive integer $s$ such that: if the pointer is outside an object but is still within $s$ display units from any edge of the object, that object can still be selected. See “Selection and Deselection Ranges,” below.

**Deselection range**
A positive integer $d$ such that: if a mouse button is clicked (for example, to initiate a move, edit, or copy operation) when the distance between the pointer and a selected object is greater than $d$ display units, then the selected object is deselected.

The deselection range is set by default to the largest possible value (to indicate infinity), so that a selected object is never automatically deselected.

**Edit range**
A positive integer $e$ such that: if the pointer is within $e$ display units of an edge or vertex of the selected object, then clicking the Move-Edit mouse button will execute the edit operation; otherwise it is a move operation. Two numbers are supplied: one in display units, the other in pixels. $e$ takes the value that results in a larger on-screen distance.

**Select drawn objects**
Instructs L-Edit to automatically select an object after it is created. This is useful for designers who like to position or edit objects after creating them instead of while drawing them.
Selection and Deselection Ranges

The values set for **Selection range** and **Deselection range** govern the operation of implicit selection.

When you click the MOVE-EDIT mouse button, L-Edit selects geometry within the selection range and deselects geometry outside the deselection range. In the following illustration, the selection range is bounded by a heavy black outline.

When multiple objects are within the selection range, L-Edit determines which object to select using the following priority: (1) objects the pointer is *inside*, ordered by the closest edge; (2) objects the pointer is *outside*, yet still within the selection range, ordered by the closest edge.
Setup Design—Drawing

Use the **Setup Design—Drawing** dialog to set assorted default drawing parameters.

![Setup Design—Drawing dialog](image)

**Default port text size**
Default text size, in display units, for ports.

**Nudge amount**
Default amount, in display units, by which objects are moved by the nudge operation.

**Default persistent ruler display**
See “Edit Object(s) for Rulers” on page 294.

Setup Design—Libraries

Use **Setup > Design—Libraries** to add, remove, change or redirect the libraries available to a design. When you load a design, L-Edit populates this window with the files listed in the lib.defs (see “lib.defs - the Library Path File in OpenAccess” on page 46.)

For both the Name and Path fields, a right-click in the field area opens a context menu you can use to edit the name or file path, browse to a folder or path, or set the library paths to be relative or absolute.
Library files can also be referenced when you import a GDSII file (see “Importing GDS Files” on page 133).

**Setup Design–Object Snap**

Object snapping snaps the cursor to user selected snap points on objects. Snap points include object vertices, edge midpoints, edges, wire centerlines, box and circle center points, instances, and ports. Please refer to “Aligning and Distributing Objects” (page 335).

**Setup Design—Interactive DRC**

Interactive DRC allows DRC rules to be checked in real time, while polygons are being drawn or edited. Please refer to Chapter 14, “Interactive DRC” (page 509).

**Setup Design–Node Highlighting**

Interactive DRC allows DRC rules to be checked in real time, while polygons are being drawn or edited. Please refer to Chapter 15, “Node Highlighting” (page 517).
Setup Design–Valid Vias

are those used in assisted routing and SDL. Please refer to “Valid Vias” on page 396.

Layer Setup

L-Edit supports an unlimited number of technology layers, as *layer purpose pairs*. A layer purpose pair is a specific layer purpose combination identified either by layer name and purpose name, or layer number and purpose number. A layer purpose pair must have a GDSII number in order to be fabricated.

To add to or edit the layer structure in the active file, use the menu option **Setup > Layers**, right-click from either of the layer palettes, or double-click anywhere on the Compact Layer palette to launch the **Setup Layers** dialog.

The **Layers** list on the left shows all the defined layers in the active file. Note that no two layers can have the same name. Also, derived layers must be positioned in the list below the layers from which they are derived. When you are using an OpenAccess database, the layer name is followed by a colon and the purpose name.

The order of layers in **Setup Layers** set the default layer order in all other layer lists.
Common Controls in the Setup Layers Tabs

There are three tabs in the Setup Layers dialog: General, Derivation, and Rendering. The following controls are available in each of these tabs:

**Add**
To add a layer to the list, click the **Add** button. A New Layer \([n]\) (where \([n]\) is the number of the new layer) will be added to the layer list; this name can be edited by clicking **Rename**.

**Delete**
To delete a layer, highlight the layer in the list and click **Delete**. A layer can only be deleted if it contains no geometry.

**Rename**
Opens a dialog that lets you change the layer name and OpenAccess layer number.

**Copy**
To add a copy of an existing layer, highlight the layer in the list and click **Copy**. The new layer is placed below the selected layer, with “Copy of” preceding the layer name.

**Note:** GDSII number, GDSII data type, and CIF name are not copied to the new layer.

**Move Layer**
Click on the up or down arrow to reposition the highlighted layer in the list.

**Properties**
Opens the standard Properties dialog, where you can define and attach any number of properties to the layer. See

**Purposes**
A user-defined value describing the layer.
Chapter 2: Application and Design Setup

Layer Setup

**General Layer Parameters**

The **General** tab allows you to set the following layer properties:

<table>
<thead>
<tr>
<th>General</th>
<th>Derivation</th>
<th>Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locked</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hidden</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protected</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Locked**

When this box is checked, geometry on a layer cannot be drawn, moved, edited or deleted.

**Hidden**

When this box is checked, the layer is not displayed in layout.

**Protected**

When this box is checked, objects on a layer cannot be selected.

**Valid**

L-Edit does not require this attribute, but it reads and retains the valid/invalid status of a layer-purpose pair for compatibility with other applications.

The **Valid** checkbox controls whether a layer is displayed in the layer palette list. If it is checked, the layer is displayed in the layer palette list. If it is not checked, the layer is not displayed in list.

However, when you choose “All” in the filter field of the layer palette, both valid and invalid layers will be displayed in the list (see “Using the Layer Palette” on page 196.)

**Layer-to-substrate capacitance**

Specifies the **Area** capacitance between the layer and the substrate (in aF/sq. micron), and the **Fringe** capacitance (in fF/micron).

**Resistivity**

Specify the resistivity (resistance per square unit area) of the layer material in Ohms/square.
Import/Export

To edit the import/export parameters of the selected layer, enter values in these fields as appropriate:

- **GDSII number**—an integer that indicates the GDSII layer number
- **GDSII data type**—an integer that can be used in combination with the GDSII number to identify an additional layer. For further information, see “GDSII Data Type” on page 161.
- **CIF name**

Default wire settings

Set the default **Width** (in display units), **End style**, **Join style** and **Miter Angle** for the layer’s wire settings. See “End Styles and Join Styles” on page 297 for more information on wire settings.

Derivation Layer Parameters

The **Derivation** tab allows you to define new layers that are derived from existing ones using logical and selective operations. The various settings are discussed in “Generating Derived Layers” on page 458.
Rendering Layer Parameters

The options on the **Rendering** tab control layer appearance.

A layer’s appearance is determined by when it is drawn with respect to the other layers in the design—its **pass** value, and whether its color is added, subtracted, or replaces the colors of the layers that are drawn before it—its **mode**.

For each layer, you can also specify a color and a pattern for the fill and outline of drawn elements in normal state and in selected state. The elements for which color and pattern can be set are:

- Object
- Selected object
- Port box
- Selected port box
- Port text
- Selected port text
- Wire centerline
- Selected wire centerline
**Note:** Wire centerlines, whether or not the wire is selected, are always rendered in a 1 pixel wide solid pattern.

### Mode

Use **Mode** to control how a layer affects the appearance of the layers with which it overlaps. Options are:

- **Add**—use a logical OR operation
- **Subtract**—use a logical AND NOT operation
- **Paint**—use a logical OVERWRITE operation

[See “Rendering Modes” on page 94 for more information.](#)

### Pass

Use **Pass** to control the order in which layers are rendered. Pass values range from 1 to 10, where 1 is rendered first and 10 is rendered last. (See “Rendering Passes” on page 97 for more information.)

### Paint background color on previous pass

Use this option to properly render stacked vias. When this box is checked, the layout background color clears all layers with a pass value less than that of the active layer prior to rendering of a patterned object.

This option is available only for non-selected objects with a non-solid fill, for layers with a pass value greater than one.

### Pattern (for Fill and Outline)

Select one of the predefined patterns from the drop-down list or use **Other** to create one of your own. (See “Pattern” on page 97 for more information.)

- **None** fills in none of the pixel elements used to create a pattern, **Solid** fills them all in.

### Color (for Fill and Outline)

To set rendering color, click one of the bars or bit codes in the drop-down list. (See “Color” on page 98.)

(The number of colors available and how they are sorted is controlled in the **Setup Colors** dialog. See “Setting the Number of Available Colors” on page 60 for more information.)

### Style (for Outline only)

Click on the ellipsis (…) to set the outline style for the selected element. L-Edit opens the **Outline Style** dialog where you can specify a line style, line width, and the line width unit of measure. (See “Outline Style” on page 99.)

### Rendering Modes

L-Edit objects can be drawn in one of three modes that control how layers are rendered, particularly when they overlap, **Paint**, **Add**, or **Subtract**.
Drawing mode applies to fill, or to outline if the layer has no fill. If a layer has fill and outline, the outline is always rendered in **Paint** mode, no matter which mode is selected for the fill. Objects that are selected are always rendered in **Paint** mode and cannot be set to another mode.

**Paint Mode**

Object colors and patterns are combined as layers are drawn. Overlapping objects produce entirely new colors and patterns. This mechanism ensures that regions of overlap are displayed in a meaningful way without obscuring the presence of other objects.

In **Paint** mode, the color of regions of overlap and all drawn objects is determined by a logical OVERWRITE operation. The bit values of a layer always overwrite the bit values of the layers drawn before it (i.e. with equal or lower **Pass** values). For layers with the same pass value, rendering proceeds in layer list order.

However, if the **Paint background color on previous pass** checkbox is enabled, a layer is rendered in two passes. The first pass clears the layer with a pass value lower than that of the active layer, and replaces it with the background color of the layout window. The second pass then draws the stipple pattern for the layer as usual. (This option is only available to layers with a non-solid fill pattern and a pass value greater than one.)

For example, in the illustration below, both via layers are set to paint the background on previous layer passes. Layer **Via 2** (black stipple pattern C, with a pass value higher than the others) is rendered over the layer **Poly** (red, with a pass value lower than Via 2) and over Via 1 (blue stipple pattern R, with a pass value lower than via 2) layers.
Add Mode

In Add mode, the color of regions of overlap is determined by a logical OR between the bits of the color code for each layer. The bit values of an Add layer are added to the bit values of those layers drawn before it.

For example, if Metal1 is blue with a Pass value of 1:

`[Image of Metal1 layer]`

and Metal2 is grey with a Pass value of 2:

`[Image of Metal2 layer]`

their overlapping areas will be rendered as a darker blue, as shown in the figures below:

`[Image of overlapping Metal1 and Metal2]`

The region of Metal1 and Metal2 overlap is shown by a third darker color created by the logical OR (Add) operation.

Subtract Mode

In Subtract mode, the color of regions of overlap is determined by a logical AND NOT. The complement of the bit values of a Subtract layer are subtracted from the bit values of those layers drawn before it. (This corresponds to the “clear” mode in previous version of L-Edit.)

Note that a subtract layer “clears” colors rendered before it but has no effect on the appearance of objects on layers rendered after it.

For example, if Metal1 is [Image of Metal1 layer] with a Pass value of 1 and Add mode,

P Select is [Image of P Select layer] with a Pass value of 2 and Add mode, and

Via1 is [Image of Via1 layer] with a Pass value of 2 and Subtract mode, then

objects on Via1 will be rendered as [Image of rendered Via1], as shown in the following illustration.
To define vias that appear transparent, the via layer Pass value must be higher than that of the metal layers (2 or higher), so that the via layer color operation will affect them, or the via layer should come after the metal layers in the layer list. The mode should be Subtract and the color 1111 (black). With a color bit code of 1111, subtract mode yields the logical operation AND NOT of 1111—or the logical AND of 0000, the complement of 1111. The logical AND of 0000 and any other color will be 0000, so the drawn color will always be 0000.

A box on Via1 layer has the same color everywhere it overlaps other layers, because its bits are subtracting all previously rendered bits.

![Diagram showing via layer rendering]

**Rendering Passes**

Each layer is rendered in one pass. The order in which layers are rendered is determined first by position in the layer list and then by the pass value. Lower pass values are rendered first, with possible values ranging from one to ten.

You can set your rendering passes so that objects will be drawn in a way that parallels the manufacturing processes or simply to control rendering of overlaps. However, a layer that is derived from other layers must be below all its source layers in the layer list.

For layers with both fill and an outline, fill is rendered first, then the outline, and both will be completed before the fill for the next layer is started. Selection is always rendered with the last pass (10).

Note that for rendering, pass order will take precedence over layer order in the Layers list.

**Pattern**

To change the stipple pattern, select one of the predefined patterns from the drop-down list. None fills in none of the pixel elements used to create a pattern—a None pattern for fill yields no fill, a None pattern for outline yields no outline. A Solid fills in all pixel elements.
You can also pick **Other** to open the **Create Pattern** dialog, where you can design a new pattern.

New patterns are added to the bottom of the drop-down list with the label “custom.”

**Color**

To set rendering color, click one of the bars or bit codes in the drop-down list.

The number of colors available and how they are sorted is controlled in the **Setup Colors** dialog. (See “Setting the Number of Available Colors” on page 60 for more information.)
Outline Style

You can set both a line style and width for outlines. Line width can be measured in either pixels or display units.

To outline an object when it is selected, click on the “Selected...” element name in the list, define an outline style, and set the fill pattern to None.

For curved geometry, outline thickness is displayed at a fixed value of 1 pixel.

Note that boxes and rectangular polygons are rendered so that all edges include the snap grid pixels. When two such drawn objects coincide, they will therefore overlap by a width of one pixel. Such an overlap will be rendered in a distinct color, creating a visible line, unless you use a layer setup where the outline is the same color as the fill.

Listing the Layers in a File

Use File > Layer Summary to generate reports on the layers in a design. You can specify the layer types to list, the sort order and the level of detail to include (GDSII number, data type, derivation equation, etc.)

Special Layers

Special layers are used to represent L-Edit layout elements such as the display grid, origin, drag boxes, and so on. They are treated just like other layers—you define them using Setup > Layers, draw objects on them, and can specify design rules for them. In fact, they may be identical to layers used for other
purposes (for example, Poly can be designated as the Grid layer). Choose \textbf{Setup > Special Layers} to designate special layers. Select an appropriate layer from the drop-down menu next to each field.

<table>
<thead>
<tr>
<th>Grid</th>
<th>The layer on which the displayed grid points are drawn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drag box</td>
<td>The layer on which the boxes displayed during a drag operation are drawn and on which the nibbling wire is drawn.</td>
</tr>
<tr>
<td>Origin</td>
<td>The layer on which the crosshair marker representing the coordinate system origin is drawn.</td>
</tr>
<tr>
<td>Cell outline</td>
<td>The layer on which instanced cell outlines are drawn.</td>
</tr>
<tr>
<td>Error</td>
<td>The layer on which DRC and SPR error markers are drawn.</td>
</tr>
<tr>
<td>Icon</td>
<td>The layer on which non-fabricating comment items are drawn.</td>
</tr>
</tbody>
</table>

\textbf{Layer Purpose Pairs}

L-Edit supports \textit{layer purpose pairs}. A layer purpose pair is a specific layer purpose combination, identified either by layer name and purpose name, or layer number and purpose number.

When you draw an OpenAccess object, it is saved in the design database as an object on a layer purpose pair. L-Edit references the layer name and purpose name that correspond to a layer purpose pair from the technology file.

The layer number corresponding to a given layer name cannot be changed. Similarly, the purpose number corresponding to a given purpose name cannot be changed. However, an object can be changed from one layer-purpose pair to another.

\textit{Predefined Layer Purposes}

A set of common layer purpose pairs are predefined in L-Edit.

- redundant -8
- gapFill -7
- annotation -6
- OPCAntiSerif -5
- OPCSerif -4
slot -3
critical 224
soCritical 225
soError 226
ackWarn 227
info 228
track 229
blockage 230
grid 231
warning 234
tool1 235
tool0 236
label 237
flight 238
error 239
annotate 240
drawing1 241
drawing2 242
drawing3 243
drawing4 244
drawing5 245
drawing6 246
drawing7 247
drawing8 248
drawing9 249
boundary 250
pin 251
net 253
cell 254
all 255
Predefined Layer Name-Number Mappings

A set of layer name-number mappings are predefined in L-Edit.

<table>
<thead>
<tr>
<th>Layer Name</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unrouted</td>
<td>200</td>
</tr>
<tr>
<td>Row</td>
<td>201</td>
</tr>
<tr>
<td>Group</td>
<td>202</td>
</tr>
<tr>
<td>Cannotoccupy</td>
<td>203</td>
</tr>
<tr>
<td>Canplace</td>
<td>204</td>
</tr>
<tr>
<td>hardfence</td>
<td>205</td>
</tr>
<tr>
<td>softfence</td>
<td>206</td>
</tr>
<tr>
<td>y0</td>
<td>207</td>
</tr>
<tr>
<td>y1</td>
<td>208</td>
</tr>
<tr>
<td>y2</td>
<td>209</td>
</tr>
<tr>
<td>y3</td>
<td>210</td>
</tr>
<tr>
<td>y4</td>
<td>211</td>
</tr>
<tr>
<td>y5</td>
<td>212</td>
</tr>
<tr>
<td>y6</td>
<td>213</td>
</tr>
<tr>
<td>y7</td>
<td>214</td>
</tr>
<tr>
<td>y8</td>
<td>215</td>
</tr>
<tr>
<td>y9</td>
<td>216</td>
</tr>
<tr>
<td>designFlow</td>
<td>217</td>
</tr>
<tr>
<td>stretch</td>
<td>218</td>
</tr>
<tr>
<td>edgeLayer</td>
<td>219</td>
</tr>
<tr>
<td>changedLayer</td>
<td>220</td>
</tr>
<tr>
<td>unset</td>
<td>221</td>
</tr>
<tr>
<td>unknown</td>
<td>222</td>
</tr>
<tr>
<td>spike</td>
<td>223</td>
</tr>
<tr>
<td>hiz</td>
<td>224</td>
</tr>
<tr>
<td>resist</td>
<td>225</td>
</tr>
<tr>
<td>drive</td>
<td>226</td>
</tr>
<tr>
<td>supply</td>
<td>227</td>
</tr>
<tr>
<td>wire</td>
<td>228</td>
</tr>
<tr>
<td>pin</td>
<td>229</td>
</tr>
<tr>
<td>text</td>
<td>230</td>
</tr>
<tr>
<td>device</td>
<td>231</td>
</tr>
</tbody>
</table>
In addition to the predefined purposes, you can create custom purposes. When you load an OpenAccess database, it will often have custom purposes already defined.

Use the **Purposes** button in **Setup Layers** or **Setup > Custom Layer Purposes** to open the **Custom Layer Purposes** dialog. You use the same dialog to view existing custom layer purposes and to define new ones. To add a new layer purpose, simply enter a name and number in the first blank row.

### Adding a Custom Purpose

Use the **Purposes** button in **Setup Layers** or **Setup > Custom Layer Purposes** to open the **Custom Layer Purposes** dialog. You use the same dialog to view existing custom layer purposes and to define new ones. To add a new layer purpose, simply enter a name and number in the first blank row.

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>dummy</td>
<td>10</td>
</tr>
<tr>
<td>keep_out</td>
<td>16</td>
</tr>
<tr>
<td>doc</td>
<td>27</td>
</tr>
<tr>
<td>lvs</td>
<td>28</td>
</tr>
<tr>
<td>mdp</td>
<td>29</td>
</tr>
<tr>
<td>opc</td>
<td>30</td>
</tr>
<tr>
<td>no_dummy</td>
<td>35</td>
</tr>
<tr>
<td>shv</td>
<td>40</td>
</tr>
<tr>
<td>iv_op</td>
<td>41</td>
</tr>
<tr>
<td>res</td>
<td>42</td>
</tr>
<tr>
<td>cap</td>
<td>43</td>
</tr>
<tr>
<td>do</td>
<td>44</td>
</tr>
</tbody>
</table>

- **border**: 232
- **snap**: 233
- **align**: 234
- **prBoundary**: 235
- **instance**: 236
- **annotate**: 237
- **marker**: 238
- **select**: 239
- **grid**: 251
- **axis**: 252
- **hilite**: 253
- **background**: 254
Adding, Copying or Renaming a Layer

When you Add or Copy a layer purpose pair, you must create a unique layer and purpose pair. You can add a new layer purpose pair that uses an existing layer name with a different purpose, or you can enter a new layer name and select a purpose. When you enter a new layer name you may also enter a layerOpenAccess number. If you use an existing name, L-Edit will display the appropriate value, disabled, in the Number field. When you rename a layer you can only change the name, all its other attributes remain.

Deleting a Layer

Use the Delete button in Setup Layers to delete the layer-purpose pair that is highlighted in the list of layers. If there is no geometry on that layer-purpose pair, the layer-purpose pair will be deleted instantly with no messages.
If there are objects on a layer-purpose pair that you are trying to delete, L-Edit will give you the option to continue even if the layer contains geometry. When you delete a layer that has objects, those objects remain attached to their layer-purpose pair although the layer definition is removed.

A check in this box means delete the geometry on the layer within the specified scope and the layer definition itself.

No check means delete the layer definition. If the layer contains geometry, the layer will be automatically redefined without attributes; the objects will remain.

If you elect to delete a layer even if it does contain geometry, you can limit the checking that L-Edit performs to just those cells you have reserved, or all cells in the design.

- **Reserved cells** - checks and deletes the layer only in the cells you have reserved.
- **All cells** - Using the standard evaluation criteria (see “Setting the Scope of Action for Multiple Cell Operations” on page 260, this option checks all cells in the design hierarchy and performs the layer deletion on those for which a reservation can be acquired. Because this option forces L-Edit to load all cells in the design this process can be slow.
3 Opening and Closing Designs

Creating a New Design

Use File > New > Design to start a new design. You can choose between a single file TDB database for a single user, or a multiple file OpenAccess database for multiple users, plus a technology reference for either type of database.

**Database**

For OpenAccess, enter a **Name** for your file and the **Path** of the location where it will be saved. For a TDB file you enter the path and name in the same field.

**Technology Reference**

Please refer to the section “Establishing Technology Information” on page 48. If you do not select a technology reference, the new design is opened with the default L-Edit **empty** settings of a white background, basic application layers and no DRC rules.
If you create a new OpenAccess design, L-Edit will prompt you to select a top-level library; and also to select the libraries to reference (see “File Contents in an OpenAccess Database” on page 47.)

Opening an Existing Design

Use one of the following operations to open a file. Note that all Tanner Tools Open dialogs have a drop-down menu of supported file types to help limit the display when you are browsing.
Select **File > Open** or press **Ctrl+O**; click the open file button ( ); drag and drop a file icon from Windows explorer or open a recently used file by selecting its name from the bottom of the **File** menu.

Look in

The source directory.

File name

The name of the file to open. You can use the wildcard character (*) to narrow down the list of available files. Only one file can be opened at a time.
Chapter 3: Opening and Closing Designs

Saving a TDB File

L-Edit saves the following information for each TDB file:

- Size and location of all layout windows opened in this file
- Which cells are open
- The following viewing options:
  - Zoom levels of open cells
  - Visibility of origin and grid
  - Visibility of arrays and ports
  - Last view
  - Visibility of layers

You have several options when saving a TDB file in L-Edit:

### Files of type
Limits the search to just one file type. Predefined file types in the drop-down menu are:

- Tanner Database Files (*.tdb)
- OpenAccess Database Files (*.oa, lib.defs)
- Tanner Standard Extract Files (*.ext)
- Netlist Files (*.tpr; *.ed)
- Design Rule Command Files (*.drc)
- Cross-Section Process Definition Files (*.xst)
- Nodal Capacitance Files (*.cap)
- Standard Delay Format Files (*.sdf)
- Application Configuration Files (*.ini)
- Error Files (*.err)
- Tanner Text Files (*.txt)
- UPI Macro (*.c, *.h, *.cpp, *.cc, *.hpp, *.cxx, *.hxx)
- SPICE Netlist (*.spc, *.sp, *.cir)
- Verilog (*.v, *.va, *.verilog)
- Assura Command file (*.a)
- Tanner Extract File (*.a)
- Html or Text (*.htm, *.html, *.txt)
- All Files (*.*)

### Open as read-only
When checked, opens a design file so that changes to the file cannot be saved. Note that this read-only option is only supported for TDB files.
- Save All - use File > Save All or press Ctrl+S to save a file using its current filename and path.
- Save Cell to TDB file - see “Saving a Single Cell to Another File” on page 233.
- Save As - to rename or relocate a TDB file. See “Renaming or Relocating a TDB File” on page 113.
- Save Copy - to save a TDB file as an OpenAccess database. See “Converting a TDB File to OpenAccess Format,” below.

**Converting a TDB File to OpenAccess Format**

With the TDB file open, use File > Save Copy > OpenAccess Database to convert a TDB file to the OpenAccess database format.

The entire design hierarchy will be copied, converted and saved to the location you specify. The TDB file as a whole becomes the top-level library when it is converted.

You will need to specify a path where the design will be saved and the type of OpenAccess file system encoding, which determines how case is handled when creating cell names on disk.

Note that when there are Xref cells in the TDB file, the top-level .tdb must be in the same folder as the technology setup files.

![Save Copy As OpenAccess Database](image)

**Save copies of all TDB libraries – translating to OA format – to folder:**

Browse to or select the folder where the new OpenAccess database will be saved.
Chapter 3: Opening and Closing Designs

Saving a TDB File

The Four TDB to Open Access Conversion Options

Local

Saving a TDB file to an OpenAccess database with the local option performs the conversion with no mapping of layer names.

All technology settings including layer names, rendering patterns, layer setup, Via Templates, Node Highlighting, Interactive DRC, and Standard DRC setup will be preserved unchanged.

Attached or Referenced

Saving a TDB file to an OpenAccess database with the Attached or Referenced option performs the conversion and establishes an OpenAccess Attached or Referenced link to a technology library.

Layers in the TDB file are mapped to layers in the technology library by use of a map file. The map file provides a mapping from the GDS number/datatype in the TDB file to the layer-purpose-pair names in the OpenAccess tech library. In order to successfully complete the mapping, all layers in the TDB file that have objects on them should i) have a GDS number/datatype in Setup Layers, ii) have a mapping from GDS number/datatype to layer-purpose names in the map file, and iii) have a corresponding layer-purpose pair in the OpenAccess tech library.

Rendering patterns in the TDB will be preserved in the mapped layers for the Referenced option, but for the Attached option, layer rendering patterns will come from the library. For both, technology setup from the original TDB (other than rendering) will be lost, so it is the responsibility of the Attached or Referenced library to have the needed technology setup. It is therefore important to have a technology library with a complete technology setup. In particular, via templates in the TDB file should have corresponding via templates with the same name in the technology library before saving TDB to

Technology

Sets how the technology file for the new OpenAccess database will be acquired. Please refer to “The Four TDB to Open Access Conversion Options,” below, for more details.

- Local – technology information will be read and translated from the TDB file and stored with the new OpenAccess files.
- Attached – attaches a technology file and disallows any other technology, even a local or top-level library, to be referenced (though the attached library may have its own references.)
- Referenced – allows you to use any set of referenced and local technology files.
- Copied – creates a copy of the full design and the entire library hierarchy in the specified directory.

Use technology from OA library path:

Browse to or select the folder containing the technology file (lib.defs) you want to use.

Map layer names based on GDS numbers in GDS layer map:

Check this box to replace Tanner TDB layer names with a set of layer names from the specified layer map file.

L-Edit will use GDS numbers to perform the mapping of GDS layer and datatype to layer purpose pairs.

If this box is not checked, L-Edit will retain the layer names from the tdb file.
OpenAccess with Attached/Referenced option, as via templates from the TDB will be discarded and those from the library will be used.

Note that if parameters of a via template in the TDB are different to the corresponding parameters in the library, placements of such via templates will change, as the library parameters will be used after the conversion. If a via template in the TDB does not have a matching via template in the technology library, then all placements of this via will be lost when the TDB is saved to OpenAccess. Since the attached or referenced technology library is not modified by the Save TDB to OpenAccess operation, the technology library may easily be exchanged when an update is available.

**Copied**

Saving a TDB file to OpenAccess with the Copied option will convert the TDB file to an OpenAccess database, and will map layers to the layer names and layer numbers used in the technology library.

The resulting OpenAccess database is self-contained – no Attach or Reference is made to the technology library. Via templates that are defined in both the TDB and the technology library will take their parameters from the technology library. Rendering patterns and layer derivations from the original TDB file will be preserved. The OpenAccess database resulting from the conversion contains a merging of data from the original TDB file and the technology library.
Renaming or Relocating a TDB File

To save a TDB file to a different name or location, choose File > Save As.

Save in
The target directory.

File name
The name under which the file corresponding to the active window is to be saved. The space above this field lists all files of the specified type in the target directory. If you choose a name that already belongs to an existing file, L-Edit prompts you for permission to overwrite the existing file.

Save as type
By default, the active file is saved as its current type. A text file can be saved with any extension. You can use Save as type to revert to an earlier L-Edit version format for backwards compatibility.
### Converting an OpenAccess File to TDB Format

To save an OpenAccess file as a TDB file, use **File > Save Copy > Tanner Database**. L-Edit saves each of the libraries in the design as a separate TDB file that contains all required technology information.

![Save Copy as TDB](image)

### Renaming or Relocating an Open Access File

To save a copy of an OpenAccess library to a different name or location, choose **File > Save Copy > OpenAccess Database**. In the **Save Copy** dialog you can enter a new path and folder name in the field or click on the browse button to search for a target destination.

Note that you cannot save a copy to an existing folder; OpenAccess requires a new folder for the copied library.

![Save Copy](image)

### Closing a Design

To close the current file, choose **File > Close** or press **Ctrl+W**. If a file contains any unsaved changes, L-Edit will prompt you to save them.

![L-Edit](image)

- **Yes** saves the file. L-Edit saves all cells, unreserves all cells, and closes the design. If the file is new, the **Save As** dialog appears (see “Saving a TDB File,” below).
- **No** unreserves all cells and closes the design without saving any changes
- **Cancel** closes the dialog and returns to the editing session.
Closing a design silently releases reserved cells that have not been modified. (See “Reserving a Cell” on page 224.)

If there are new cells that have been created but not saved, when a design is closed their folders will be removed from disk. Similarly, when you exit L-Edit entirely, L-Edit will prompt for unsaved changes in all open designs.

Printing

Use **File > Print** or press **Ctrl+P** to print. For TDB files, L-Edit opens the following dialog:

![Print dialog](image)

- **Name**: The device to which the data in the active file will be printed.
- **Properties**: Opens the **Printer Properties** dialog.
- **Print to file**: Prints to a file instead of the physical printer. If this option is checked, clicking **OK** opens the **Print to File** dialog.
- **Print range**: The set of pages to be printed.
- **Copies**: The number of copies to be printed, with the option to collate.
- **Layout area**: Drop-down menu that gives the option to print
  - **Entire cell**
  - **Current window only**
Resolution

Printer resolution in dots per inch (dpi).

- **Best** (matches printer resolution)
- **600 x 600**
- **300 x 300**
- **150 x 150**
- **75 x 75**

Scaling

Option buttons control the magnification of the layout on the printed page.

- **Pages**—the number of pages (width \(\times\) height) that the printed layout will occupy. For example, a 3\(\times\)2 page scale will result in a layout that spans 6 printed pages: 3 wide and 2 high. Multiple pages can be pasted together after printing to create the layout as on the screen.

- **Display Units per**—the ratio of display units to physical units on the page. (To specify display units, use the pull-down menu in the locator bar.) Options are **Inch** and **centimeter**. Selecting this option will activate the field where you specify the number of display units per the specified physical unit.

Options

Opens the **Layout Print Options** dialog (see “Layout Print Options,” below).

For text files, the dialog appears without the options for **Layout area**, **Resolution** and **Scaling**.

![Print dialog](image)
### Layout Print Options

This dialog lets you control print legends, margins and page overlaps, as well as how much print information will be stored in the L-Edit internal buffer.

#### Include header

Lets you select whether a header will be printed (at the top left of the plot) and what it will contain. Possible values are:

- \n — new line
- %c — cell name
- %d — current date and time
- %n — file name
- %p — fine path
- %s — scale
- %v — L-Edit version
- \ — back slash
- % — percent sign

#### Include legend

Toggles printing of a default legend for the layers shown in the plot, positioned below the header.

#### Include ruler

Toggles printing of vertical and horizontal rulers showing major and minor tick marks, along the image edge.

#### Coordinates at corners only

Toggles printing of tick mark values at the corners of the image only (checked) or along the entire length of the rulers.
| **Reverse black and white (if background is black)** | Allows you to invert black and white rendering in the plot (as compared to the onscreen rendering) for layouts with a black background. Note that only true black (red=0, green=0, blue=0) values will be reversed to white. All other colors will be printed as they are rendered onscreen. |
| **Margins** | Controls the margin sizes for single page printing. |
| **Multi-page margins** | Controls the margin sizes for multi-page printing. |
| **Multi-page overlap** | Controls the amount of image overlap on each page during multi-page printing. |
| **Memory** | Sets the size of the internal L-Edit memory buffer for bitmap information sent to the printer, in megabytes. L-Edit prepares the entire bitmap that represents a plot in computer memory and then sends it to the printer driver. Most printer drivers limit the size of such a bitmap, but not many can accurately measure what this limit should be. Due to this limit, when printing large plots, L-Edit splits the bitmap into sections and sends them one by one to the printer driver. L-Edit uses the “bitmap buffer” value to calculate how many sections it needs. The bigger the value the fewer sections needed and the faster L-Edit will print in almost all cases. 8 MB is the best value for most printers, but some plotters benefit from a higher value. Some older printers can handle only 1MB sections. |
Print Setup

To set up printing parameters, choose File > Print Setup. For TDB files, the following dialog appears:

(For text files, the dialog appears without the Layout area, Resolution, and Scaling options.)

**Printer**

The default device to which the active text file will be printed. The drop-down list shows available printers.

**Properties**

Opens the Printer Properties dialog.

**Paper**

Specifies the paper size and source.

**Orientation**

Specifies the orientation of the printed page as portrait or landscape.

**Layout area**

Drop-down menu that gives the option to print the entire cell or just that part of the cell displayed on your monitor.

**Resolution**

Printer resolution in dots per inch (dpi).

- **Best** (matches printer resolution)
- **600 x 600**
- **300 x 300**
- **150 x 150**
- **75 x 75**
Scaling  

Option buttons control the default magnification of the layout on the printed page.

- **Pages**—the number of pages (width × height) that the printed layout will occupy. For example, a 3×2 page scale will result in a layout that spans 6 printed pages: 3 wide and 2 high. Multiple pages can be pasted together after printing to create the layout as on the screen.

- **Display Units per**—the ratio of display units to physical units on the page. (To specify display units, use the pull-down menu in the locator bar.) Options are **Inch** and **centimeter**. Selecting this option will activate the field where you specify the number of display units per the specified physical unit.

**Options**  

Opens the **Layout Print Options** dialog (see “Layout Print Options” on page 117).

**Network**  

Opens the **Connect to Printer** dialog, which allows you to connect to printers in your shared environment.

---

**Print Preview**

**File > Print Preview** displays the active cell as it will be printed in full-page mode. Cells cannot be edited in **Print Preview**.

- **Print**  
  Opens the **Print** dialog (see “Printing” on page 115).

- **Next Page**  
  Displays the next page in the window.

- **Prev Page**  
  Displays the previous page in the window.

- **Two Page**  
  Displays two pages in the window.

- **Zoom In**  
  Magnifies the scale of the display.

- **Zoom Out**  
  Reduces the scale of the display.

- **Close**  
  Closes the **Print Preview** dialog.
In addition to layouts, you can also preview text or a Design Navigator display.

**Copying and Saving Layout Images**

To copy an image of your layout, use **File > Image > Copy to Clipboard**. L-Edit copies an image of the active layout window, including the origin and grids if they are displayed, to the clipboard.

If you want to save the layout image to a file, choose **File > Image > Save to File**. L-Edit saves a .bmp format image to the name and location you enter, using the cell name as a default file name. After you press the Browse button you can change the file format to .jpg/.jpeg, .tif, .gif or .png - .bmp is simply the default.

To set the image size and resolution before saving it to a file, choose **File > Image > Save to File with High Resolution**. L-Edit saves the image to the name and location you enter, using the cell name as a default file name.

**Active window dimensions** Shows the original dimensions of the layout image.

**Custom dimensions** These fields allow you to set the size of the window when the image is saved – sets the background or “canvas” size; does not scale the image itself.

**Use screen resolution** Shows the original resolution of the layout image.

**Use custom resolution** This field allows you to set the resolution of the image when saved.
File Information

Access information about the current file by choosing **File > Info.**

**File path**
The path and filename of the active file.

**File info**
Includes **Author**, **Organization**, and **Information** (notes or messages) for the active document. **Information** can contain a maximum of 256 characters. **Object Count** shows the total number of polygons in a file. (You can also use “Mark Cells for Flattening” (page 528) to determine this value at any time.)

**File flags**
Locks the current file. Locked files cannot be edited, but data from the file can be copied to another file.

**File dates and versions**
The date and time the file was created and last revised. **Setup info** and **Layout info** include version numbers. The version numbering system provides an internal accounting method for tracking layout design and file setup changes. Major numbers are increased by clicking the **Increment** button; minor numbers are automatically incremented each time changes in the file are saved.

**Properties**
Accesses the **Properties** dialog. For more information on file properties, see “Properties” on page 123.
Listing Object Types by Layer

Use **File > File Object Summary** to generate a text file listing the number of shapes (boxes, polygons, wires, circles, pie wedges and tori), ports, layers and persistent rulers on each layer in a file.

Transferring File Information to Cells

Use **Tools > Add-Ins > Transfer File Info to Cells** to copy just the author and organization information from the file information to the information dialog for all cells in the file.

Properties

L-Edit properties are comprised of a name and value and are used to store information. You can attach properties to L-Edit objects, layers, instances, cells, and files.

When you attach a property to a cell or instance, it is local to that cell or instance only. Properties attached to primitive cells do not propagate throughout the design hierarchy.

If you copy a cell, however, the copied cell **will** contain any properties created in the original.

Properties are classified by type and can be arranged in hierarchical groups. Properties have many applications, including:

- Netlist extraction—you can use properties to control the information extracted from a layout.
- Design management—in conjunction with a UPI macro, you can use properties for such tasks as counting the instances of a cell.
- Design documentation—you can load a text file or other document into a property attached to a file, cell, layer, or other object.
Property Types

Valid property data types include the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Property without a value—often used simply to create a level of hierarchy.</td>
</tr>
<tr>
<td>Integer</td>
<td>Signed integer from -2,147,483,648–2,147,483,647.</td>
</tr>
<tr>
<td>Byte</td>
<td>Unsigned integer from 0–255.</td>
</tr>
<tr>
<td>Real</td>
<td>Floating point (8-byte) number from 1.7E +/- 308.</td>
</tr>
<tr>
<td>String</td>
<td>Alphanumeric string.</td>
</tr>
<tr>
<td>Logical</td>
<td>Boolean with values True or False.</td>
</tr>
<tr>
<td>Pointer</td>
<td>Address of a location in RAM memory.</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary large object.</td>
</tr>
</tbody>
</table>

Binary Large Object ("Blob") Properties

When you select or create a property of the type binary large object (BLOB), the Load and Save buttons below the sub-properties list are activated.

- **Load**: Accesses a standard Windows file browser in which you navigate to the object to be loaded into a BLOB property.
- **Save**: Accesses a standard Windows file browser in which you specify the filename and path of the file to which the BLOB will be written. The default filename extension is .blo.

Viewing and Editing Properties

Properties are viewed and edited from the Properties dialog, which you access as follows. For a layout object, select the object, choose **Edit > Edit Object(s)**. For a layer, use the menu command **Setup > Layers**. For a cell, highlight the cell in the Cell list and use the command **Cell > Info**. For a library, highlight the library in the Libraries list and use the menu command **File > Info**.
From any of these dialogs, use the **Properties** button to open the **Properties** dialog:

The Properties tree on the left displays the properties for the selected file, cell, layer, or object. Properties without sub-properties are displayed with rectangular white icon, those with a sub-property are displayed with a folder icon. L-Edit uses a period (.) to separate levels of hierarchy in the Properties tree. When a property is selected, its name, type, and value are displayed in the fields to the right.

### Adding a Property

To add a property, select a property in the Properties tree that you want to be the parent for the new property and click **Add Property**, right-click and select **Add property** from the pop-up menu or press the Insert key (Ins). L-Edit displays the **Add Property** dialog:

<table>
<thead>
<tr>
<th>Property name</th>
<th>Full name of the new property.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property type</td>
<td>Data type of the new property (see “Property Types” on page 124).</td>
</tr>
<tr>
<td>Property value</td>
<td>Value of the new property. Type the value in this field or click the arrow to select from a list of previously used values.</td>
</tr>
</tbody>
</table>
Deleting a Property

To delete a property, select it in the Properties tree and either click **Delete Property**, right-click and select **Delete property** from the pop-up menu, or press the Delete key (Del).

Renaming a Property

To rename a property, right-click on it in the Properties tree and select **Rename Property** (shortcut F2) from the pop-up menu.

Deleting a Property Value

To delete a property value, select the icon in the Properties tree or select the sub-property in the **sub-properties** list and click the **Delete Value**, or use the right-click menu.

Editing a Property Value

When you select a property in the Properties tree, you can edit its value in the **Value** field. You can also edit values for sub-properties of the selected property in the **Value** column of the **sub-properties** list. Select an item in the **sub-properties** list and press F2. Alternatively, you can right-click the item and select **Modify value** in the resulting pop-up menu.

**Adding a Copyright, Logo or Text to a File**

L-Edit provides several methods for adding text to your layout. A “layout generator” feature lets you add true type fonts using any of the fonts installed on your PC by rendering characters as bitmaps.

You can add text, a company name or a logo by instancing standardized cells from a special file that is shipped with L-Edit, and there is also a UPI command “**LCell_MakeLogo**” (page 1279).
Adding True Type Fonts to a Design

Use Draw > Layout Generators > Convert Formatted Text to Layout to add text in any of the fonts available on your PC. Text is created as layout geometry by producing a box corresponding to each pixel of a character after it has been rendered as a bitmap.

Layer
Select the layer on which the text geometry will be drawn.

Text size
Enter a size for the text. You may need to experiment with this value to get the text size you prefer as it will be relative to the scale of your design.

Note: Point size is not an absolute value but differs for every font, as it is a value based upon the distance between lines of set type and the difference between the highest ascender and lowest descender in a given font. One convention is to specify the height of the capital M as 72% of the font height. This works fairly well for most Roman style fonts but is not as consistent for other font styles.

Font
Click this button to select from the available fonts.

Bold and italic styles are supported when available. The Font dialog shows a preset font size of 16, but it is the Text Size field on the Text to Layout dialog that controls text size.

Text
Enter the text string you want to add as layout.

Merge resulting polygons
When this checkbox is empty, which is the default, L-Edit will merge rows of contiguous pixels to reduce file size and processing time. When the box is checked to turn the option on, the resulting layout can be merged into non-contiguous polygons of any shape.

Snap polygons to manufacturing grid
Check this box to snap the text polygons to the manufacturing grid when placed.
Using the \texttt{alphabet.tdb} File

The \texttt{Draw > Layout Generators > Layout Text Generator} command takes cells from \texttt{alphabet.tdb} and adds them as layout geometry to your current design.

For tdb designs with libraries, the “Library Name” should be set to the top level library and disabled, as is done for “Group,” “New Cell,” and other commands that create cells. See also “\texttt{LCell\_MakeLogo}” (page 1279).

\texttt{alphabet.tdb} is a special file containing cell instances for each letter in the alphabet, numerals and special characters, plus cells for a copyright and a logo. It is shipped with L-Edit in the \texttt{C:\Program Files\Tanner EDA\Tanner Tools v xx.yy\AddIns} directory.

![Logo Example]

See also the UPI macro “\texttt{Logo Generator}” (page 1881).
Adding a Copyright, Logo or Text to a File

**Layout Text Generator**

- **Layout Text String**: Enter your text here. A maximum of 1024 characters is allowed. You can use the following formatting codes:
  - &c - Center text
  - &l - Align text left
  - &r - Align text right
  - \n - Insert new line
  - &d - Insert current date
  - &t - Insert company logo
  - &m - Insert mask copyright

- **Layer Name**: The layer on which to generate the text. The layer must exist in the current technology setup.

- **Print Date**: Prints the current date.

- **Print Mask Copyright**: Prints the copyright symbol from the **Mask Copyright Symbol** cell in the **alphabet.tdb** file.

- **Text Size**: Enter the desired text size in the current display units. You may need to experiment with this value to get the text size you prefer as it will be relative to the scale of your design.

  In **alphabet.tbd**, though the height of MBB of each character is 1 micron, the characters of that block capital alphabet do not span that full height. So, if you enter a text size of 1 in a design using microns as the display unit, the text itself will not be a full micron high. The extra space is needed for the ascenders and descenders in lowercase and non-block lettering.

- **Underline**: Underlines the text string and, if used, date. This option is useful when you need to etch away the material underneath the text.
Rescaling a Design

A design scaling wizard, Setup > Rescale Wizard... is available to assist in rescaling a design, either by specifying a current and a new feature size, or by specifying a percentage to scale the design. In the first step of the Rescale Wizard you specify how much you want to rescale the design.

In the second step of the Rescale Wizard you specify which quantities you want to rescale. You also choose the method used to rescale the design. Rescaling by changing the database resolution will rescale all geometry and settings, with no truncation. Rescaling without changing the database resolution can cause truncation as geometry coordinates and settings will be multiplied by the ratio of
From feature size/To feature size. Quantities that will be truncated are indicated in the Rescale wizard dialog.
Exiting L-Edit

Choose **File > Exit** to close a design. If there are unsaved changes, L-Edit prompts you like to save those changes.

Closing a design silently releases reserved cells that have not been modified. If there are new cells that have been created but not saved, when a design is closed their folders will be removed from disk.

Similarly, when you exit L-Edit entirely, L-Edit will prompt for unsaved changes in all open designs.

**Abnormal Termination**

When L-Edit is launched, a process called `oaSFLockD.exe` starts automatically. When L-Edit is closed or terminated normally, this process continues to run.

When L-Edit terminates abnormally (because of a crash, because the operating system closes, etc.) while cells are reserved, the lock files for those cells will remain on disk.

If another user attempts to lock one of those cells, L-Edit uses `oaSFLockD.exe` to check for the process identity of the user who had the cells reserved. If it is not present, L-Edit will allow the new user to overwrite the reservation created by the first user.

However, if both `oaSFLockD.exe` and L-Edit are terminated abnormally (aborting from Task Manager, due to power failure, etc.), then the .cdslock files belonging to the first user will need to be manually deleted, since the `oaSFLockD` process is no longer present to tell other users that the lock can be overwritten.
4 Importing and Exporting Files

Importing Files

You can import GDSII, CIF, DXF and Gerber format files, as well as the bitmap formats GIF, JPEG, TIFF and BMP into an L-Edit file. You can also import a Cadence Virtuoso® technology file or a Silicon Canvas Laker file into L-Edit.

**File > Import Mask Data** can be used to bring mask information into an active file or to establish a new file.

Importing GDS Files

Use **File > Import Mask Data > GDSII** to import GDSII files into L-Edit.

![Import GDSII dialog box](image)

- **From file**
  - Name of the file containing the design data to be imported. L-Edit will search for zipped GDS files with a `.gds.gz` extension as well as `.gds` files, and will unzip a file if needed during the import procedure.

- **If Unknown Layers are found**
  - If GDS layer numbers are found that are not in the setup file then **Prompt** to ask how to map the layer, or automatically **generate new layers**. See “Prompt if unknown layers are found” on page 134.

- **Treat different GDSII data types on a layer as different layers**
  - Overwrites existing cells:
    - **All**
    - **Top design only**
    - **None**

- **Database resolution**
  - **From GDSII file:**
    - microns
  - **Custom:**
    - microns
### Prompt if unknown layers are found

If you selected to prompt for unknown layers then L-Edit will display the dialog shown below if your design has a layer for which there is no corresponding layer in the setup file.

When a design contains a name/purpose layer with no GDS number/data type, L-Edit uses the **Setup > Layers** map to assign matching layers GDS parameters. When a matching layer purpose pair contains one GDS parameter but the other one is empty L-Edit will assign the missing value. If a
layer/purpose pair already contains GDS parameters they will not be changed. The log in the command window will indicate that all layer mappings were imported.

After importing a GDSII file, L-Edit produces a log file summarizing settings, showing status, and providing detailed warning and error messages.

**Setting the View Level**

Often you will want to set or change the hierarchy view level when importing a GDSII file. Because this setting is saved with the L-Edit design file for each cell view, it cannot be controlled using the technology setup information. You can instead use a read-only registry string to set the Hierarchy Level value for all new cells created during GDSII import. This is an optional setting with a persistent value. Set it to the integer representing the desired level, or use zero to set it to “all.”

For example, if your GDSII file shows all levels but you are importing into L-Edit to place or position cells, you would use:

```
[HKEY_CURRENT_USER\Software\Tanner EDA\L-Edit\Import/Export\GDSII]
 ''HierarchyLevelOnGDSImport''="1"
```

...to view just the cell names and rotation at the first level of the view hierarchy.
Importing CIF Files

Use File > Import Mask Data > CIF to import CIF files into L-Edit. Note that to avoid errors, rotated boxes should be entered as polygons.

- **From file**: Name of the file containing the design data to be imported.
- **If unknown layers are found**: If CIF layer names are found that are not in the setup file, choose either **Prompt** to ask how to map the layer, or **Generate new layers** to automatically create those layers.
- **Read rectangular polygons as boxes**: Rectangular polygons in CIF will be converted to boxes in L-Edit if this option is checked. This option speeds processing, as boxes consume less memory and are drawn faster than polygons.
- **Overwrite existing cells**: Choose to overwrite **All** cells, those in the **Top design only**, or **None**.
Importing OASIS Files

Use File > Import Mask Data > OASIS to import GDSII files into L-Edit.

**From file**
Name of the file containing the design data to be imported. L-Edit will search for files with extension `.oas` as well as `.oasis`.

**If unknown layers are found**
If layer numbers are found that are not in the setup file then prompt to ask how to map the layer, or automatically generate new layers.

**Treat different GDS data types on a layer as different layers**
If this option is checked, layers with the same layer number but different data types will be placed on different layers. The layers will have the same number but different names and different data types. If an object has a combination of layer number and data type that corresponds to an existing layer in a TDB setup file, L-Edit maps an object to that layer. If a layer does not exist corresponding to the layer number and data type, or if you are importing into an empty setup, a new layer will be created and the object will be mapped to that layer.

If this option is not selected, layers with the same layer number but different data types will be placed on the same layer — L-Edit will map to the first layer with the corresponding layer number and will ignore the data type. If a layer with the layer number does not exist, L-Edit will create it.

**Overwrite existing cells**
Select which cells will be overwritten; All, those from the highest level library Top design only, or None.
Chapter 4: Importing and Exporting Files

Importing Files

Prompt if unknown layers are found

If you selected to prompt for unknown layers then L-Edit will display the dialog shown below if your design has a layer for which there is no corresponding layer in the setup file. The log in the command window will indicate that all layer mappings were imported.

- **Database resolution:** When a setup file is specified, this field displays the database resolution of the setup file which is used for import and may not be modified. (The Custom option can only be used when you are importing using an empty setup file.) If the resolution in the setup file is different from the resolution in the OASIS file, then the OASIS data is scaled to maintain the physical size of the layout.

- **Import:** Imports the specified file.

**Generate new layer for this GDSII number and data type combination**

- Creates a new layer for one specific unknown value.

**Generate new layers for all unknown GDSII layer numbers**

- Generates a new layer for all unknown values

**Map to layer:**

- Maps the unknown value to an existing layer selected from the pull-down menu.

After importing a OASIS file, L-Edit produces a log file summarizing settings, showing status, and providing detailed warning and error messages.
Importing DXF Files

Use **File > Import Mask Data > DXF** to import DXF files into L-Edit. Note that a cell must be active to initiate this command.

L-Edit will attempt to read the specified DXF file into a new L-Edit file. The technology settings for the new file are taken from the L-Edit file open at the time the import is executed. L-Edit generates a log file detailing conversion issues.

DXF ARCs are imported as zero-area curved polygons; DXF LINEs and open POLYLINES are imported as wires of zero width. If the open POLYLINE has curved segments, the wire will contain a multi-segment approximation to the curved edge, not exceeding 256 segments.

**From file**
Name of the file containing the design data to be imported.

**Scale**
DXF files do not contain explicit scaling information. Thus, you must specify how many display units each DXF unit corresponds to.

**Objects with Non-Zero Elevation**
DXF files can contain 3-D data.

- **Ignore these objects** is selected, only objects with Z=0 coordinates will be read in; this filter is applied on a vertex-by-vertex basis.
- **Collapse these objects to Z=0 plane** is selected, the Z value of each vertex is ignored.
- **Accept only objects in Z range** is selected, the Z range of elements to be imported can be specified. This option helps import 3-D data from tools like SolidWorks. (Prior to version 15.14, L-Edit created the same objects at the min-Z and max-Z for the 3-D object, making reconstruction of polygons from collections of lines practically impossible given the duplicate objects.) Please see “Draw > Convert > Connect Segments,” below for an illustration.
Importing Files

Chapter 4: Importing and Exporting Files

Importing Files

Draw > Convert > Connect Segments

Use this command to fix unresolved polygons, typically after a DXF file is imported into L-Edit.

Geometry in a DXF design that is drawn using polylines is replaced with zero-width wires when imported to L-Edit. When L-Edit finds a closed path during import, the individual segments are replaced by a single polygon. If the “Merge Open Polyline” option is enabled (by entering a value in the “Merge with Tolerance” field), L-Edit will search for segments having endpoints within the specified tolerance, and try to build closed paths of such segments. If a closed path is found, the individual segments are replaced by a single L-Edit polygon. See “Draw > Convert > Connect Segments,” below.

**Note:** The search for possible closed paths is **not** exhaustive, so selecting as small a “Merge with tolerance” value as possible is the best strategy.

**Do not import hatch fills or dimension**

When checked, omits hatch fills and dimensions from the information imported to L-Edit.

As of version 15.14, when you elect to ignore hatches, L-Edit silently ignores the DXF HATCH element. Previously, they were imported hatches and L-Edit deleted cells whose names began with *D or *X (as AutoCAD creates subcells that contain the hatch, but other tools do not.)

**Do not import visible attributes as ports**

When checked, visible attributes in the DXF file will not be considered ports in L-Edit.

**Draw > Convert > Connect Segments**

Use this command to fix unresolved polygons, typically after a DXF file is imported into L-Edit.

Geometry in a DXF design that is drawn using polylines is replaced with zero-width wires when imported to L-Edit. When L-Edit finds a closed path during import, the individual segments are replaced by a single polygon. If the “Merge Open Polyline” option is enabled (by entering a value in the “Merge with Tolerance” field), L-Edit will search for segments having endpoints within the specified tolerance, and try to build closed paths of such segments when it imports an open polyline in from DXF format.

For this reason, you will usually need to enable the “Merge with Tolerance” field, and will probably also the “Collapse these objects to Z=0 plane” option to collapse objects with non-zero elevation to the Z=0 plane when you first import a DXF file into L-Edit.

Occasionally, depending upon the original drawing and the merge tolerance setting during import, some polylines intended to circumscribe a polygon may remain unconnected in L-Edit. In this case you should select the unconnected zero-width wires, select the layer on which the polygon is to be drawn, and use **Draw > Convert > Connect Segments** to specify the largest gap between objects that will be joined in order to form a polygon.
When polylines are closer together than the value in the **Tolerance** field of **Connect Segments**, they will be connected to form a polygon on the selected layer.

![Before and After Geometry](image)

The illustration below shows before and after geometry, and the values used in the import dialog.

![Before and After Geometry](image)

This polygon with an MBB of (-0.5, -0.5) %, 5) DXF units using a “Scale” value of 1000 microns per DXF units, was imported with “Collapse these objects to Z=0 plane” and both “Do not import hatch fills or dimension” and “Do not import visible attributes as ports” enabled.
Importing Gerber Files

Use **File > Import Mask Data > Gerber** to import Gerber files into L-Edit. L-Edit reads “extended Gerber,” also known as RS-274X. Note that a cell must be active to initiate this command.

### Import Gerber

**From file**

Gerber file(s) to import. Choose **Single file** or **Multiple files in folder** to import multilayer Gerber data where each layer is in a separate file.

**Default layer**

Objects that do not have a layer specified in the input file will be added to this layer.

**Extension**

When you are importing from multiple files, you can leave this field blank to import all layer files, or use it to indicate just one extension type L-Edit should import.

**Default format**

Gerber uses strings of digits to encode numbers where the location of the decimal point is implied (for example, the string 0150 could mean 0.15, 1.5 or 15.) Also, trailing or leading zeros can be suppressed.

The **Default Format** fields tell L-Edit how to interpret these strings. It is not necessary to set this data in modern Gerber files, because they use the %FS% format specifier directive (with L, T, A, I, X and Y options). If necessary, you can specify the correct number of digits to the left and right of the decimal point in the **digits left of decimal** and **digits right of decimal** fields.

### Importing Graphics

You can use L-Edit to import a GIF, JPEG, TIFF or BMP format raster graphic into your design file. L-Edit converts the bitmap into layout geometry by drawing a corresponding box for each pixel in the image.
Chapter 4: Importing and Exporting Files

Importing and Dithering Options

Import options include dithering, for photographs or other images with many colors or high resolution, and merging, which reduces the number of polygons that are drawn, as shown below.

Use the menu command **Draw > Layout Generators > Import Image** to set your import options.

**Import Image to Layout**

- **File**: Name of the file to import. Valid formats are GIF, JPEG, TIFF and BMP.
- **Layer**: Select the layer on which the image geometry will be drawn.
- **Pixel size**: L-Edit draws a square corresponding to each pixel of the imported bitmap. Use this field to enter the dimension of that square. For example, if the pixel size is set to 1um and the image is 594 pixels wide, then the resulting layout will be 594um long. This field is initialized to the manufacturing grid.
Exporting Files

You can export GDSII, CIF, DXF and Gerber format files from an L-Edit file.

Exporting GDS Files

After exporting a GDSII file, L-Edit produces a log file with detailed warning and error messages. However, L-Edit does not create backups for GDSII files. If you try to write to an existing GDSII file, L-Edit will warn that you are about to overwrite a file.

There is a set of UPI commands you can use to retain information that is only persistent in the TDB file, that control GDS export settings such as “Selected cell and its hierarchy” and “Do not export hidden objects and layers.” Please see “LGDSParam” (page 1738), “LGDSExport Scope” (page 1741), “LGDSExportLogParams” (page 1745).

Note that generator cells are never exported, as they are always empty.
Mapping the GDS Data Type and Purpose

L-Edit accepts and preserves non-standard GDSII numbers when importing and exporting GDSII files. For further information on assigning and propagating GDSII data types, see “Assigning Data Types” on page 162 and “GDS Layer Map File Format” on page 58.

L-Edit assigns a number to each layer in the design in order to conform to GDSII syntax. To modify a GDSII layer number prior to exporting the file, use Setup > Layers—General. Select a layer in the Layers list and enter the appropriate value in the GDSII number field.

L-Edit supports referencing a layer map file for mapping layer-purpose pairs to GDS numbers and datatypes that you can use instead of the mappings defined in Setup Layers. The layer map file has four columns:

<table>
<thead>
<tr>
<th>Layer Name</th>
<th>Layer Purpose</th>
<th>GDS Layer Number</th>
<th>GDS Datatype</th>
</tr>
</thead>
</table>

Both the map file and Setup Layers allow multiple layer-purpose pairs to map to the same GDS layer. However, unlike the mappings in Setup Layers, with a map file a single layer-purpose pair can be output to multiple GDS layer numbers.

Polygons with Too Many Vertices

Because GDSII files do not contain curves, L-Edit converts circles, pie wedges, curved-sided polygons and tori to straight-sided polygons when GDS mask data is exported, with vertices snapped to the manufacturing grid. Please see “Converting Curved and other Objects to Polygons” on page 347 and “Snapping Existing Geometry to the Manufacturing Grid” on page 305 for details on these processes.

However, the GDSII limit for a list of vertices is 8192 (with the first vertex repeated as the last vertex), and due to limitations in some GDS tools and foundry specification there is a traditional best practice limit of 200 vertices.

If your design contains a wire or polygon with more than 200 vertices, L-Edit will write a warning to the GDS Export log.

To resolve this problem, you can fracture a polygon having a large numbers of vertices into many polygons with fewer vertices using Draw > Convert > Fracture Polygons. Note that the fracture operation does not modify wires, circles, pie wedges, or tori. (See “Fracturing Polygons” on page 350.)
Use **File > Export Mask Data > GDSII** to export GDSII files from L-Edit.

### Export GDSII

**To file**
Name of the file to which you want to export GDSII.

**Zip output file**
Check this box to zip the file during export. L-Edit will append .gz to the file extension.

**Export scope**
- **All cells** - Exports all cells from all open libraries to the GDSII file.
- **Active cell** - Exports just the active cell.
- **Fabrication cell** - Exports just the fabrication cell.
- **Cell**: Use the drop-down menu to select a specific cell from any of the open libraries to export.
- **Cells from libraries**: Use this option to export one or more entire libraries (separated with a comma and space) from those that are open.
- **Cells selected in navigator** - Exports just the cells highlighted in the Libraries navigator.

**Include hierarchy**
Exports just the specified cell and all cells instanced in it, and instantiated in those cells, etc. to the base of the cell's hierarchy. Applies to any option except “All cells.”
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclude libraries</td>
<td>If <strong>Include hierarchy</strong> is checked, you can use this field to exclude one or more libraries (separated with a comma and space) from the exported file. References to cells in excluded libraries are written as links.</td>
</tr>
<tr>
<td>Save log file to</td>
<td>Enter a file name and directory location for the export log file.</td>
</tr>
<tr>
<td>Open log in window</td>
<td>When this option is checked, the log file will open in the L-Edit text editor.</td>
</tr>
<tr>
<td>GDSII units: default (1 database unit = 0.001 microns)</td>
<td>When selected, L-Edit converts object dimensions into units of 0.001 micron (the default GDSII database unit) when exporting a GDSII file. For example, a $10 \times 10$ box with 1 internal unit = 1 lambda = 1 micron in the L-Edit layout would be recorded in the GDSII file as having dimensions of 10,000$\times$10,000 database units.</td>
</tr>
<tr>
<td>GDSII units: Custom</td>
<td>Check this option and enter a value to convert object dimensions to database units of other than 0.001 microns, or to a specified number of user units when exporting a GDSII file.</td>
</tr>
<tr>
<td>Cell names</td>
<td>Select <strong>Preserve Case</strong> to leave the case of cell names unmodified, <strong>Upper Case</strong> to convert cell names to upper case or <strong>Lower Case</strong> to convert cell names to lower case (note that some GDSII systems do not recognize lowercase letters).</td>
</tr>
<tr>
<td>Restrict cell names to</td>
<td>Select <strong>32 characters</strong> to conform to the GDS standard. Cell names will be truncated to 32 characters and if needed, modified to avoid name collisions, in which case a warning will be written to the GDSII export log. Select <strong>128 characters</strong> to conform to Cadence Virtuoso® GDS import capability, where truncation will occur at 128 characters. Select <strong>Unlimited</strong> for no truncation.</td>
</tr>
<tr>
<td>Map file</td>
<td>Use this field to specify the path and name of a layer map file from which L-Edit will read layer-purpose names to GDS number and datatype mapping, ignoring those defined in the Setup Layers dialog.</td>
</tr>
<tr>
<td>Do not export hidden objects</td>
<td>When this option is checked, L-Edit will not write any objects that are hidden either by object type or by layer. When this option is off, all objects will be written to the GDSII file, whether hidden or not.</td>
</tr>
<tr>
<td>Overwrite object data type with layer data type</td>
<td>When this box is checked, an object will be written to the GDSII file with the data type of the layer on which it is drawn. If the layer does not have a data type, the object retains its data type. For further information, see “Assigning Data Types” on page 162.</td>
</tr>
<tr>
<td>Calculate MOSIS Checksum</td>
<td>Calculate checksum required for submitting GDSII files to MOSIS.</td>
</tr>
<tr>
<td>Check for self-intersecting polygons and wires</td>
<td>When this box is checked, polygons and wires will be checked for self intersections, and self intersections will be reported to the GDS export log. This option can significantly slow down time required for export.</td>
</tr>
<tr>
<td>Fracture polygons with more than n vertices</td>
<td>Polygons with more than $n$ vertices will be split into multiple polygons with less than or equal to $n$ vertices. Circles, Arcs, and Tori will also be fractured if their vertex count exceeds $n$ when they are converted to polygons for GDS export.</td>
</tr>
</tbody>
</table>
Exporting CIF Files

L-Edit cannot export layers without legal CIF names. L-Edit does not create backup files for CIF files. If you try to write to an existing CIF file, L-Edit will warn that you are about to overwrite a file.

CIF files support circles, therefore they are not approximated during the export process. Curves, however, are not supported. If your design contains curves they are automatically approximated during export.

Tanner reads either CIF extensions 85 or 91 to attach the instance name to a cell, but writes 91.

Use File > Export Mask Data > CIF to export CIF files from L-Edit.

Exporting OASIS Files

After exporting an OASIS file, L-Edit produces a log file with detailed warning and error messages.
Use **File > Export Mask Data > OASIS** to export OASIS files from L-Edit.

**Export OASIS**

- **To file**: Name of the file to which you want to export OASIS.
- **Zip output file**: Check this box to zip the file during export. L-Edit will append .gz to the file extension.
- **Export scope**
  - **All cells**: Exports all cells from all open libraries to the OASIS file.
  - **Active cell**: Exports just the active cell.
  - **Fabrication cell**: Exports just the fabrication cell.
  - **Cell**: Use the drop-down menu to select a specific cell from any of the open libraries to export.
  - **Cells from libraries**: Use this option to export one or more entire libraries (separated with a comma and space) from those that are open.
  - **Cells selected in navigator**: Exports just the cells highlighted in the Libraries navigator.
- **Include hierarchy**: Exports just the specified cell and all cells instanced in it, and instantiated in those cells, etc. to the base of the cell's hierarchy. Applies to any option except “All cells.”
**Exclude libraries**
If **Include hierarchy** is checked, you can use this field to exclude one or more libraries (separated with a comma and space) from the exported file. References to cells in excluded libraries are written as links.

**Save log file to**
Enter a file name and directory location for the export log file.

**Open log in window**
When this option is checked, the log file will open in the L-Edit text editor.

**GDSII units: default (1 database unit = 0.001 micron)**
When selected, L-Edit converts object dimensions into units of 0.001 micron (the default GDSII database unit) when exporting a GDSII file. For example, a 10×10 box with 1 internal unit = 1 lambda = 1 micron in the L-Edit layout would be recorded in the GDSII file as having dimensions of 10,000×10,000 database units.

**GDSII units: Custom**
Check this option and enter a value to convert object dimensions to database units of other than 0.001 microns, or to a specified number of user units when exporting a GDSII file.

**Cell names**
Select **Preserve Case** to leave the case of cell names unmodified, **Upper Case** to convert cell names to upper case or **Lower Case** to convert cell names to lower case (note that some GDSII systems do not recognize lowercase letters).

**Map file**
Use this field to specify the path and name of a layer map file from which L-Edit will read layer-purpose names to GDS number and datatype mapping, ignoring those defined in the Setup Layers dialog.

**Do not export hidden objects**
When this option is checked, L-Edit will not write any objects that are hidden either by object type or by layer. When this option is off, all objects will be written to the OASIS file, whether hidden or not.

**Overwrite object data type with layer data type**
When this box is checked, an object will be written to the OASIS file with the data type of the layer on which it is drawn. If the layer does not have a data type, the object retains its data type. For further information, see “Assigning Data Types” on page 162.

**Calculate MOSIS Checksum**
Calculate checksum required for submitting GDSII files to MOSIS.

**Check for self-intersecting polygons and wires**
When this box is checked, polygons and wires will be checked for self intersections, and self intersections will be reported to the OASIS export log. This option can significantly slow down time required for export.

---

**Exporting DXF Files**

Use **File > Export Mask Data > DXF** to export DXF files from L-Edit. L-Edit will attempt to create legal DXF names for cells and layers by replacing illegal characters with "_". DXF export writes out all layers that are actually used in the design including those with Special Layer designation. However, hidden layers are not exported.
If the Design Navigator is in focus when you initiate DXF export, it will run on the selection in the Design Navigator as well as on the active layout window.

**Export DXF**

- **To file**: Name of the file to which you want to export DXF.
- **Export L-Edit wires as DXF open polylines (ignores end and join styles)**: When this option is checked, all wire is exported as an “open polyline” (the DXF name for a path), which is simply a series of vertices and a width. End style and join style information is lost. Otherwise, the wire is converted to a polygon, and a “closed polyline” (the DXF name for a polygon) is written out. This shape accurately reflects the outline of the original L-Edit wire.
- **Export curved objects as straight polygons**: When this option is checked, circles and curved objects will be exported as polygons, with the vertices of the polygons on the manufacturing grid. Otherwise, curved objects are exported to DXF as true circles and curves.
- **Flatten output**: When this option is checked, data will be flatten during export to the DXF file.
Exporting Gerber Files

Use **File > Export Mask Data > Gerber** to export Gerber files. L-Edit will replace illegal characters such as “/”.

**To file**
Name of the file to which you want to export.

**Export polygons as**
Select one of these fill options:

- Use **RS274x (G36/G37)** for versions of Gerber that support polygons.
- Use **Window trace** to produce fill in a concentric pattern following the outline of a shape.
- Use **Raster trace** to fill from left to right.

**Units and Format**
Specify **Inches** or **Millimeters** as the unit of measure.

If necessary, set the location of the decimal point explicitly by entering a whole number in the **Digits to the left of decimal point** and **Digits to the right of decimal point** fields.

**Separate layers into individual files.**
When you choose this option, L-Edit writes out a file for each layer, applying an incrementally numbered filename of the format `{output file name}.layern.gbr`.

**Export NC Drill file**
Check this option to create an NC Drill format file and use the pull-down menu to select the **Drill layer** on which drill holes data is encoded.
Exporting PostScript Masks

The Tools > Add-Ins > PostScript Mask Separation command generates individual PostScript files from the layout database. The masks are one-to-one scale, and are either positive or negative polarity.

All layers that have a MaskPolarity string property set are written out. If the value of this property is Negative, a negative mask is written; otherwise, a positive mask is written. The user geometry is centered on the mask. The mask size, if given, sets the clipping boundary for the generated PostScript. If not set, the entire page is used (an A4/8.5" by 11" page size is assumed).

CIF File Formatting

Caltech Intermediate Form (CIF) is an ASCII file format for the interchange of mask geometry information among IC designers and foundries. CIF is defined in Introduction to VLSI Systems by Mead and Conway (Addison-Wesley, 1980). CIF files are typically saved with the .cif extension.

A CIF file may contain a single design or a library of designs. CIF assumes a right-handed geometry, with the x-axis increasing to the right and the y-axis increasing upward. The basic unit of measurement is 0.01 micron.

Commands may be used to scale object sizes, use different layers, and change the placement of objects. Comments may be added to a CIF file by enclosing them in parentheses. All CIF commands and comments must be terminated with semicolons.

L-Edit reads either CIF extension 85 or 91 to attach instance names to cells, and writes 91.

Symbols

CIF symbols are defined with the DS and DF commands. DS begins a symbol definition:

DS nnn a b;

where nnn is the symbol number and a and b are the (optional) scaling factors. All commands that follow the DS command and precede the DF command are included in the symbol. CIF symbols are always given numeric names.

The optional scaling factors a and b are applied to the integer coordinates and distances within a symbol by multiplying each value by a and then dividing the result by b. Scaling helps to shorten the length of CIF files by eliminating trailing zeros. By default, coordinates and distances in CIF are specified in units of 0.01 micron; a = 100 and b = 1 would allow values to be specified in microns instead. The
coordinates (10,6) with \( a = 100 \) and \( b = 1 \), for example, are equivalent to (1000,600) with \( a = 1 \) and \( b = 1 \). If \( a \) and \( b \) are not specified, then they are both assumed to be 1, and all integers are mapped to the 0.01 micron standard.

The **DF** command ends the last open **DS** command:

```
DF;   (end of symbol definition);
```

If no symbol is open when a **DF** command is encountered, then a warning message is generated.

Symbols may be instanced within other symbols and are functionally equivalent to L-Edit cells.

**Calls (Instances)**

Once a symbol is defined, it may be instanced with the **C** (call) command. In addition to instancing the named symbol, the **C** command also permits a variety of optional transformations to be applied:

```
C integer transformation;
```

where *integer* is the number of the symbol being called and *transformation* is an optional transformation. A transformation may be composed of several translations, mirrors, or rotations. Combinations of transformation operations are unambiguously applied from left to right as they are encountered within the command. Great care should be exercised when determining the order of transformation operations since the commutative property does not hold.

The **translation** operation specifies a coordinate. The coordinate represents the endpoint of a vector originating at (0,0). For example:

```
C 55 T -100,10;  (call command with translation);
```

calls symbol 55 and translates it 100 units in the negative \( x \) direction and 10 units in the positive \( y \) direction.

The **mirroring** operations, **MX** and **MY**, correspond to multiplying the \( x \) and \( y \) coordinates by \(-1\), respectively. For example:

```
C 99 MX;   (call symbol 99 and flip horizontally);
C 22 MY;   (call symbol 22 and flip vertically);
```

The **rotation** operation rotates the called symbol in the specified direction. Direction is indicated by a direction vector: a coordinate whose vector from the origin (0,0) sets the angle to which the symbol’s \( x \)-axis is rotated. Only the direction of the vector is significant; the magnitude is ignored. For example:

```
C 44 R 0,1;   (call command with rotation);
```

calls symbol 44 and rotates its \( x \)-axis by 90°.

**Geometric Primitives**

CIF provides commands for creating four types of geometric primitives: boxes, polygons, roundflashes (circles), and wires.

The **B** (box) command defines a rectangular box of fixed length and width. The center coordinates locate the box, and a direction vector indicates its orientation. For example:

```
B 25 60 80,40 -20,20;   (box command);
```
describes a box of length 25 and width 60, with center at (80,40) and direction vector (–20,20). The length of the box is parallel to the direction vector, and its width is perpendicular to the direction.

The \textbf{P} (polygon) command defines a polygon with a certain number of sides and vertices. \textbf{P} accepts a path of coordinates and creates the enclosed polygonal region in the order in which the vertices are specified (the edge connecting the last vertex with the first is implied). For example:

\texttt{P 0,0 0,40 20,40 20,20 40,20 40,0;}

describes an L-shaped polygon with vertices at (0,0), (0,40), (20,40), (20,20), (40,20), and (40,0).

\begin{tikzpicture}
\draw (0,0) -- (20,0) -- (20,20) -- (40,20) -- (40,0) -- cycle;
\end{tikzpicture}

To convert rectangular polygons to boxes when reading CIF into L-Edit, you must check the \textit{Read rectangular polygons as boxes} option in the \textbf{Import CIF} dialog.

The \textbf{R} (roundflash) command defines a roundflash (circle) of fixed diameter and position. For example:

\texttt{R 100 -200,350; (roundflash command);}

describes a circle of diameter 100 with center at (–200,350).

\begin{tikzpicture}
\draw (0,0) circle (50);
\end{tikzpicture}

The \textbf{W} (wire) command defines a wire with fixed width along a specified path. A wire can be described as a long run of uniform width; ideally, the locus of points within one-half width of the given centerline or path and one-half width of the endpoints (semicircular caps). For example:

\texttt{W 40 0,0 0,100 100,100; (wire command);}
describes a wire of width 40 with centerline vertices at (0,0), (0,100), and (100,100).

Layers

All primitive geometry elements must be associated with a particular fabrication mask or technology layer. Layers are specified with the \texttt{L} (layer) command. Primitives created after an \texttt{L} command belong to that layer until the layer is reset by the next \texttt{L} command. The form of the \texttt{L} command is:

\begin{verbatim}
L shortname; (layer command);
\end{verbatim}

where \texttt{shortname} is the 1–4 character layer name. Layer names must be unique and correspond to fabrication masks being constructed. You should therefore take care that the layer names you use accord with the conventions established by your fabricator. The \texttt{General} tab of the \texttt{Setup Layers} dialog correlates CIF layer names and technology layers; the CIF names are used instead of the L-Edit layer names during the conversion of the design file into CIF. Layer names that do not conform to legal CIF syntax must be modified before saving. Layer name specifications are preserved across symbol calls.

Layer names in the setup file must agree with the layer names of CIF files read in; otherwise, the geometry information on the non-matching layers in the CIF file will be transferred to the Icon layer. Your fabricator may apply additional restrictions and extensions to the CIF standard.

Fabrication Cell

One piece of information which must be supplied to your fabricator is the name of the cell which represents the top level of your design. The fabricator will typically choose the top-level cell in your design, if it is the only such cell. However, if you do not specify this information and your fabricator has a choice about which cell to fabricate, the wrong one might be chosen.

L-Edit does not accept geometry other than CIF symbols. A CIF call (instance) to the top level of a design is achieved by choosing \texttt{Cell \to Fabricate, Fabricate} causes a CIF \texttt{C} command (or call to the selected cell) to be created at the top level, effectively identifying that cell as the cell to be fabricated. L-Edit only allows a single call outside of a symbol definition. If any rotations or transformations are embedded in this outside call, L-Edit suppresses them when the file is read.

\textbf{Warning:} Once a fabrication cell has been chosen, it will remain the fabrication cell until a new one is chosen, even if it ceases to be the top-level cell in your design. Be sure to check the fabrication cell before writing a CIF file!
Chapter 4: Importing and Exporting Files

CIF File Formatting

Restrictions

L-Edit accepts forward references (symbol calls before the symbol definitions they reference), but removes them during conversion of the design into CIF.

L-Edit does not support the CIF DD (delete symbol definition) command.

Extensions

L-Edit supports two user extensions to the basic CIF syntax. The first extension is a cell name extension of the form:

\[9 \text{cellname};\]

where \text{cellname} is the name of the currently open CIF symbol. This command can only appear within the context of an open symbol (between a DS/DF command pair). The cell name may contain spaces and must be terminated with a semicolon. Duplicate, zero-length, and null cell names are not permitted.

If a CIF file does not define cell names for CIF symbols, then L-Edit automatically assigns as the cell name the expression:

\[(DS \ nnn)\]

where \text{nnn} is the CIF symbol number. This definition is suppressed when the CIF file is written out. You should therefore avoid naming cells with this syntax, or else the name will be suppressed during CIF file conversion. L-Edit reads out-of-order cell numbers, but always orders cells by number while writing out the design in CIF.

The second user extension is a port extension of the form:

\[94 \text{portname width height center}_x \text{ center}_y \text{ layer};\]

where \text{portname} is the name of the port (label), \text{x} and \text{y} are the coordinates of the port, and \text{layer} is the name of the port’s layer. This is a relatively standard port or label user extension to CIF. However, it is not as flexible as L-Edit’s definition of a port. An L-Edit port can be a point, a line, or a box, and the text can be rotated in a variety of ways; this CIF user extension can only represent a single point, with no information on the position or rotation of the associated text. When L-Edit writes a port into a CIF file, it computes the centerpoint of the port and records this in the CIF file as the position of the port. You can preserve the box associated with the port in CIF as written by L-Edit by unchecking the Export rectangular ports as center points option in the dialog. This results in the use of nonstandard notation for ports, and other software tools may not be able to read this form of CIF.

Wires in CIF

CIF was developed at a time when masks were usually created by Gerber photoplotters. Such plotters could make wires by opening a circular aperture and moving it along a pathway. The resulting wire would therefore have rounded corners and ends. This fabrication method gave rise to the CIF specification for rounded wires. However, present-day mask making is almost entirely raster-based, and thus has a strong affinity toward orthogonal structures.

Many fabricators assume CIF wires to have extended wire end styles with mitered corners. Thus to adhere to the fabricators’ implementation of wires, all your CIF wires should be of extend end style and layout join style. Fabricators such as MOSIS and Orbit often run CIF and GDSII files through a high-end program called CATS, which is used to produce formats for specific mask-making equipment from both those layout file formats. CATS uses its own clipping algorithm for acute angle CIF wires.
and GDSII paths with a pathtype of 0 or 2. This algorithm corresponds exactly with the L-Edit wire layout join style (the default), which employs a miter length of one-half the width of the wire for wires with an acute join angle. You should check with your fabricator concerning the exact method of fabrication used for wires before using wires in your layout.

**Scaling**

Apart from the user-selectable scaling of L-Edit internal units, L-Edit incorporates an implicit scaling factor while writing CIF files. Due to the manner in which geometric objects are represented in CIF, it is necessary for L-Edit to apply an implicit multiplication factor of two to all geometry as it is written out to CIF. The reason for this scaling is that CIF represents boxes with integer length, width, and center coordinates. L-Edit, however, can create boxes with fractional center coordinates: a box of width and length 3 with lower left corner at (0,0) has its center at (1.5,1.5), for example. L-Edit circumvents this problem by multiplying all coordinates by two when writing a CIF file. The same box, after being written out to a CIF file, would have a length and width of 6 and be centered at (3,3). L-Edit incorporates this multiplication by 2 into the scaling factors recorded in the CIF file, so that when the file is read in by a CIF reader it is scaled correctly.

**GDSII File Formatting**

GDSII stream format is a binary file format for interchanging mask geometry information between different IC CAD systems. The L-Edit implementation of GDSII file reading and writing conforms to the Calma Stream Format, GDSII release 6.0, with some limitations. GDSII files are typically saved with the `.gds` extension.

A GDSII file may contain a single design or a library of designs. GDSII assumes right-handed geometry, with the x-axis increasing to the right and the y-axis increasing upward. The basic unit is set to the GDSII default (user unit = 1 micron and 1000 database units per user unit).

Most L-Edit elements have a one-to-one correspondence with elements of GDSII stream files. GDSII last access time information is not supported by L-Edit. L-Edit circles are approximated by GDSII polygons. L-Edit cell names may be modified on export to GDSII.

**GDSII Properties**

L-Edit supports and transfers GDS properties for any instance, box, polygon, wire, circles, pie wedge, and torus.
The GDSII properties of objects may be examined, edited and removed by first selecting the object(s) and using **Tools > Add-Ins > Add/Edit GDS Properties** (shortcut Alt+P).

![GDS Properties dialog box](image)

Use the **Add** button to enter and the **Edit** button to change the **Attribute** number and **Value** of a property.

![Add GDS Property dialog box](image)

The **Delete** button deletes properties that are highlighted in the list, **Delete All** deletes all properties assigned to an object.

Use **Tools > Add-Ins > Display GDS Properties** to show GDS properties in the layout for all objects in a given hierarchy. Note that this operation can only be undone individually, on separate objects.
Tools > Add-Ins > Display GDS Properties in the Current Cell behaves the same on just the active cell.

Tools > Add-Ins > Remove GDS Property Labels removes GDS property display from layout, and can only be undone on individual selected objects.

GDSII Naming

The table below shows the correspondence between L-Edit elements and their GDSII names. GDSII data types for L-Edit boxes, wires, and polygons can be viewed and edited in the Edit Object(s) dialog with Edit > Edit Object(s).

<table>
<thead>
<tr>
<th>L-Edit</th>
<th>GDSII</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Stream file</td>
</tr>
<tr>
<td>Cell Definition</td>
<td>Structure</td>
</tr>
<tr>
<td>Box</td>
<td>Boundary *</td>
</tr>
<tr>
<td><strong>Box</strong></td>
<td>Box</td>
</tr>
<tr>
<td>Polygon</td>
<td>Boundary</td>
</tr>
<tr>
<td>Wire</td>
<td>Path</td>
</tr>
<tr>
<td>Circle</td>
<td>Boundary</td>
</tr>
<tr>
<td>Instance</td>
<td>SRef</td>
</tr>
<tr>
<td>Array</td>
<td>ARef</td>
</tr>
<tr>
<td>Port</td>
<td>Text</td>
</tr>
<tr>
<td>Data type</td>
<td>Data type</td>
</tr>
</tbody>
</table>

* L-Edit boxes are written to GDSII files as 4-sided boundaries (polygons). When reading boundaries from a GDSII file, L-Edit checks each one to see if it is a 4-sided orthogonal polygon, and if so, represents it as an L-Edit box.

** GDSII boxes are not intended to be mask geometry and are generally discarded by mask-making software. If L-Edit encounters GDSII boxes while reading a GDSII file, a dialog is presented with two options: discard all GDSII boxes or convert them to L-Edit boxes (mask geometry).

GDSII allows only the following restricted set of characters in cell names. “a” … “z”, “A” … “Z”, “0” … “9”, underscore “_”, question mark “?”, and dollar sign “$”. L-Edit cell names may include a fuller set of characters, some of which would be illegal in GDSII. Therefore, L-Edit checks each cell name before writing it out to a GDSII file. If any spaces “ ” are found, L-Edit replaces them with underscores.
“_” in the GDSII file. If any other illegal characters are found, L-Edit will replace them with underscores and write a message to the GDS log.

Some GDSII systems do not recognize lower case letters in cell names. For interfacing with these systems, L-Edit provides the capability to write all cell names to a GDSII file in upper case. This option is enabled by selecting Upper case in the Cell names box in the Export GDS dialog.

GDSII Date Formats

The GDSII format allows for the year to be stored in one of three formats:

- current year (e.g. 103 representing the year 2003)
- full representation (e.g., 1999)
- last two digits of the year (e.g., 32 representing the year 2032 or 1932)

When a year is read from a GDSII file, it may need to be modified to represent the correct year, depending on which date format is used. The current year format is the default. If the last two digits of the year is detected during GDSII import, L-Edit will use the algorithm shown below to modify the date that was read.

- for years less than 60, add 2000 to the year
- for years greater than or equal to 60 or less than or equal to 1960, add 1900 to the year
- for years greater than 60, do not modify the year

This approach will handle all three date formats until the year 2060.

GDSII Shape Definition

Warning: GDSII does not contain a specification for circles. L-Edit will approximates circles using polygons; circles are not preserved when a GDSII file is written and read back in. If you plan to export to GDSII format, you should display curves using the manufacturing grid (see “Setup Design–Grid” on page 82) rather than displaying them as smooth curves.

L-Edit supports all-angle rotations of instances and 90° rotations of text.

L-Edit treats four-sided polygons as boxes. If you export, then reimport, a design that contains four-sided polygons that are orthogonally oriented rectangles, L-Edit will convert them into boxes. For the purposes of fabrication, there may be no difference between a box and its equivalent polygon.

Many different versions of GDSII readers and writers exist. Some newer versions produce elements which are not compatible with older versions of GDSII. The elements in L-Edit are confined to elements which are common to all.

GDSII Data Type

GDSII layers are identified by the GDSII number assigned to that layer or, alternately, by the combination of the assigned GDSII number and GDSII data type. You can use this data type in conjunction with the GDSII layer number to overcome the 64-layer limitation in the GDSII database.

The GDSII specification indicates that the GDSII layer number and GDSII data type should have a range of 0 to 255. However, L-Edit supports non-standard GDSII layer number and GDSII data type
values in the range of -32,768 to 32,767 for compatibility with other tools that are able to output numbers outside the 0-255 range. L-Edit will write a warning to the log file during import and export indicating that the GDSII file that was read or written does not adhere to the GDSII Stream Specification.

Assigning Data Types

Use Setup > Layers—General to assign a data type to a layer, Edit > Edit Object to assign a data type to an object, or Draw > Assign GDSII Data Types to propagate the data type for a layer to all objects on that layer, overwriting any current values, for all layers in a design. You can perform this operation for the active file or all open files.

When you assign a GDSII data type to a layer, all objects subsequently drawn on that layer will acquire that data type. If you subsequently change the data type for a layer, however, the new data type will only be applied to objects drawn after the change.

If you merge intersecting objects with different GDSII data types, L-Edit replaces their respective data type values with the data type for the layer (or 0 if no data type is set for the layer) without a warning.

Wires in GDSII

The GDSII layout format allows for three different types of wires (paths), each with a unique pathtype value:

- pathtype 0: butt ends and square corners (corresponds to L-Edit Round end style with Round join style)
- pathtype 1: round ends and round corners (corresponds to L-Edit Butt end style with Layout join style)
- pathtype 2: extended ends and square corners (corresponds to L-Edit Extend end style with Layout join style)

These GDSII pathtypes correspond directly to three of the twelve possible L-Edit wires. When reading GDSII files, L-Edit sets wire end styles and join styles to match the three GDSII pathtypes. When creating GDSII output, L-Edit assigns a GDSII pathtype according to the following table. When the end styles and join styles do not correspond exactly to a GDSII pathtype (indicated in the table with an asterisk), L-Edit will provide a warning message.

<table>
<thead>
<tr>
<th>End style</th>
<th>Join style</th>
<th>GDS II pathtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butt</td>
<td>Layout</td>
<td>0</td>
</tr>
<tr>
<td>Butt</td>
<td>Miter</td>
<td>0 *</td>
</tr>
</tbody>
</table>
Many fabricators such as MOSIS and Orbit run GDSII files through CATS (a high-end program used to produce formats for specific mask-making equipment from GDSII layout files). CATS uses its own clipping algorithm for acute angle GDSII paths with a pathtype of 0 or 2. This algorithm corresponds exactly to the L-Edit layout wire join style, the default wire join style. Layout join style employs a fixed miter length of one-half the width of the wire for wires with an acute join angle.

When you are about to use wires for the first time or you are setting up the technology files for others who may use wires, take a moment to set up the wire defaults for each layer according to whether your likely output format will be GDSII. For GDSII, use one of the three legitimate combinations of end style and join style. It is also strongly recommended that you contact your fabricator before you define the wire styles for your design to understand how they will interpret GDSII wires.

### Setting GDSII Port and Port Text Size

L-Edit uses the default port text size as a reference during import and export. On export, L-Edit calculates the ratio between the default text size and a port’s actual text size and writes that value to the GDSII file. On import, it determines the absolute text size for a given port by multiplying the default port text size by the magnification factor in the GDSII file.

To change the text size of ports in a design imported or exported from GDSII, or to decrease or increase the default port text size prior to exporting the design, use **Setup Design–Drawing** or “**Resizing Text**” (page 332).

<table>
<thead>
<tr>
<th>End style</th>
<th>Join style</th>
<th>GDS II pathtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butt</td>
<td>Round</td>
<td>0 *</td>
</tr>
<tr>
<td>Butt</td>
<td>Bevel</td>
<td>0 *</td>
</tr>
<tr>
<td>Round</td>
<td>Layout</td>
<td>1 *</td>
</tr>
<tr>
<td>Round</td>
<td>Miter</td>
<td>1</td>
</tr>
<tr>
<td>Round</td>
<td>Round</td>
<td>1</td>
</tr>
<tr>
<td>Round</td>
<td>Bevel</td>
<td>1 *</td>
</tr>
<tr>
<td>Extend</td>
<td>Layout</td>
<td>2</td>
</tr>
<tr>
<td>Extend</td>
<td>Miter</td>
<td>2 *</td>
</tr>
<tr>
<td>Extend</td>
<td>Round</td>
<td>2 *</td>
</tr>
<tr>
<td>Extend</td>
<td>Bevel</td>
<td>2 *</td>
</tr>
</tbody>
</table>
5 The L-Edit Interface

Elements of the User Interface

The L-Edit interface has the following major components:

- **Menus bar**
- **Layer Palette**
- **Mouse Button Bar**
- **Toolbars**
- **Text Editor**
- **Aerial View**
- **Design Navigator**
- **Locator Bar**
- **SDL Navigator**
- **Verification Navigator**
- **Libraries Navigator**
- **and Log Window**
- **Layout Area**
- **Status Bar**

**Menus**

L-Edit uses the following menus:
Chapter 5: The L-Edit Interface

Elements of the User Interface

The Window Menu

The Window menu lets you cascade or tile the open layout windows so you can see them all. It also lists the contents of the open windows.

The Help Menu

The Help menu has commands that open product documentation, provide links and diagnostic functions for customer support, display version information, manage commuter licensing and to install the example files that are shipped with L-Edit.

Opening the Documentation

To open the user guides and manuals, press the help button ( ) or select from the Help menu commands. L-Edit will launch Adobe Acrobat™ Reader and open the selected documentation in .pdf format.
Installing Examples

Click **Help > Setup Examples** to install the example and tutorial files that are included with L-Edit.

Managing Commuter Licensing

*A commuter license* is a portable license in a multi-seat environment that can be transferred from the system where L-Edit is installed to another computer, for example to a laptop for travel purposes. Use
**Help > Commuter Licenses > Obtain Commuter Licenses** to obtain and return commuter licenses or to check on the licensing status of L-Edit modules.
Diagnostic Tools for Customer Support

Click **Help > Support Diagnostics** to perform the system check you will need if you contact Tanner technical support.

**Help > Support Diagnostics with DLL Search** includes in the diagnostic log the location and date of L-Edit DLLs on local or networked drives.

**Help > Support on the Web** opens a web page where you can download the most recent version of the tools for which you have a current license.

**Help > Frequently Asked Questions, online** opens a web page where you can log in to access the Tanner frequently asked questions (FAQ) pages.
Displaying the Product Version or Contacting Customer Support

To determine which version of L-Edit you are using, to find contact information for the Tanner EDA Sales and Support departments, or to quickly read memory information for your computer, choose from the tabs in Help > About.

![About L-Edit](image)

Design Navigator

The Design Navigator shows all cells in a design in a hierarchical display that includes information about parent and child cells, DRC status, whether the cell is locked and similar details. You can also use it to quickly find and open a cell instance. See “The Design Navigator” on page 242 for instructions on using this tool.
Toolbars

L-Edit provides numerous toolbars to speed editing, which you can show or hide individually using the View > Toolbars command (left), or by right-clicking anywhere in the toolbar area to open the context-sensitive menu (right).

You can move L-Edit toolbars, or undock them so they can be resized. To undock a toolbar, click on one of its edges and drag it out of the toolbar area. To resize a toolbar, click and drag on an edge once it is undocked. You can keep the toolbar “floating” (undocked), you can either return it to the toolbar area as it was by double-clicking on it, or you can drag it to any docked position you prefer.

L-Edit toolbar changes when you exit the program. If you move or resize toolbars during an L-Edit session and want to return all of them to the positions they occupied at the start of the session, check Reset Toolbars in the menu shown above.

Dockable Toolbars and Views

Dockable windows are interface objects that can be attached to the edge of a fixed or main window, as well as being able to float anywhere in the Windows interface. To change the location of a dockable window, click and drag it using the titlebar. This will trigger display of shaded blue indicators with arrows pointing to where the window will be docked.
When you hover the titlebar over an indicator, a sheer blue rectangle shows where the window will be positioned.

The window will dock when you release the mouse button with the titlebar over a docking indicator.

If the window is not over a docking indicator when you release the mouse button, it will be placed as a floating window. You can resize a floating window by dragging any of its edges or corners.

A docked window can be resized, but only from the edge opposite its docked edge.
Multiple dockable windows can be grouped together. When this is the case, only one of the windows is visible. The others are shown as tabs. Click on a tab to bring a docked window to the fore and hide the others as tabs.

### Pinning A Docked View

Docked views include a pin feature that will automatically hide a docked window by minimizing it into tabs when not in use. When you click on the tab for a pinned view, the window will reopen in its docked state.
## Standard Toolbar

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="File icon" /></td>
<td>File &gt; New</td>
</tr>
<tr>
<td><img src="image" alt="File icon" /></td>
<td>File &gt; Open</td>
</tr>
<tr>
<td><img src="image" alt="File icon" /></td>
<td>File &gt; Save All</td>
</tr>
<tr>
<td><img src="image" alt="File icon" /></td>
<td>File &gt; Print</td>
</tr>
<tr>
<td><img src="image" alt="Edit icon" /></td>
<td>Edit &gt; Cut</td>
</tr>
<tr>
<td><img src="image" alt="Edit icon" /></td>
<td>Edit &gt; Copy</td>
</tr>
<tr>
<td><img src="image" alt="Edit icon" /></td>
<td>Edit &gt; Paste</td>
</tr>
<tr>
<td><img src="image" alt="Edit icon" /></td>
<td>Edit &gt; Undo</td>
</tr>
<tr>
<td><img src="image" alt="Edit icon" /></td>
<td>Edit &gt; Redo</td>
</tr>
<tr>
<td><img src="image" alt="Edit icon" /></td>
<td>Edit &gt; Edit In-Place &gt; Push Into</td>
</tr>
<tr>
<td><img src="image" alt="Edit icon" /></td>
<td>Edit &gt; Edit In-Place &gt; Pop Out</td>
</tr>
<tr>
<td><img src="image" alt="Edit icon" /></td>
<td>Edit &gt; Find</td>
</tr>
<tr>
<td><img src="image" alt="Edit icon" /></td>
<td>Edit &gt; Find Next</td>
</tr>
<tr>
<td><img src="image" alt="Edit icon" /></td>
<td>Edit &gt; Find Previous</td>
</tr>
<tr>
<td><img src="image" alt="View icon" /></td>
<td>View &gt; Goto</td>
</tr>
<tr>
<td><img src="image" alt="View icon" /></td>
<td>View &gt; Design Navigator</td>
</tr>
<tr>
<td><img src="image" alt="View icon" /></td>
<td>View &gt; Zoom &gt; Mouse</td>
</tr>
<tr>
<td><img src="image" alt="View icon" /></td>
<td>View &gt; Insides &gt; Toggle Insides</td>
</tr>
<tr>
<td><img src="image" alt="View icon" /></td>
<td>View &gt; Hierarchy Level</td>
</tr>
<tr>
<td><img src="image" alt="Cell icon" /></td>
<td>Cell &gt; Open</td>
</tr>
<tr>
<td><img src="image" alt="Cell icon" /></td>
<td>Cell &gt; Copy</td>
</tr>
<tr>
<td><img src="image" alt="Tools icon" /></td>
<td>Tools &gt; Cross-Section</td>
</tr>
<tr>
<td><img src="image" alt="Tools icon" /></td>
<td>Tools &gt; Regenerate T-Cells</td>
</tr>
<tr>
<td><img src="image" alt="Help icon" /></td>
<td>Help &gt; L-Edit User Guide</td>
</tr>
</tbody>
</table>

## Drawing Toolbar

See “Drawing Objects” on page 278 for instructions on using these commands.

<table>
<thead>
<tr>
<th>Button</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Cursor" /></td>
<td>Cursor tool</td>
</tr>
<tr>
<td><img src="image" alt="Box" /></td>
<td>Box</td>
</tr>
<tr>
<td><img src="image" alt="Polygon" /></td>
<td>90° polygon</td>
</tr>
<tr>
<td><img src="image" alt="Polygon" /></td>
<td>45° polygon</td>
</tr>
</tbody>
</table>
### Editing Toolbar

See “Drawing Objects” on page 278 for instructions on using these commands.

<table>
<thead>
<tr>
<th>Button</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="All-angle polygon" /></td>
<td>All-angle polygon</td>
</tr>
<tr>
<td><img src="image" alt="Orthogonal wire" /></td>
<td>Orthogonal wire</td>
</tr>
<tr>
<td><img src="image" alt="45° wire" /></td>
<td>45° wire</td>
</tr>
<tr>
<td><img src="image" alt="All-angle wire" /></td>
<td>All-angle wire</td>
</tr>
<tr>
<td><img src="image" alt="Wire width" /></td>
<td>Wire width</td>
</tr>
<tr>
<td><img src="image" alt="Circle" /></td>
<td>Circle</td>
</tr>
<tr>
<td><img src="image" alt="Pie Wedge" /></td>
<td>Pie Wedge</td>
</tr>
<tr>
<td><img src="image" alt="Torus" /></td>
<td>Torus</td>
</tr>
<tr>
<td><img src="image" alt="Port" /></td>
<td>Port</td>
</tr>
<tr>
<td><img src="image" alt="90° ruler" /></td>
<td>90° ruler</td>
</tr>
<tr>
<td><img src="image" alt="45° ruler" /></td>
<td>45° ruler</td>
</tr>
<tr>
<td><img src="image" alt="All-angle ruler" /></td>
<td>All-angle ruler</td>
</tr>
<tr>
<td><img src="image" alt="Instance" /></td>
<td>Instance (Cell &gt; Instance)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Edit &gt; Duplicate" /></td>
<td>Edit &gt; Duplicate</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Rotate &gt; 90 degrees" /></td>
<td>Draw &gt; Rotate &gt; 90 degrees</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Rotate &gt; Rotate" /></td>
<td>Draw &gt; Rotate &gt; Rotate</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Flip &gt; Horizontal" /></td>
<td>Draw &gt; Flip &gt; Horizontal</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Flip &gt; Vertical" /></td>
<td>Draw &gt; Flip &gt; Vertical</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Slice &gt; Horizontal" /></td>
<td>Draw &gt; Slice &gt; Horizontal</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Slice &gt; Vertical" /></td>
<td>Draw &gt; Slice &gt; Vertical</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Nibble" /></td>
<td>Draw &gt; Nibble</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Merge" /></td>
<td>Draw &gt; Merge</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Boolean Grow/Shrink Operations" /></td>
<td>Draw &gt; Boolean Grow/Shrink Operations</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Group" /></td>
<td>Draw &gt; Group</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Ungroup" /></td>
<td>Draw &gt; Ungroup</td>
</tr>
<tr>
<td><img src="image" alt="Edit &gt; Edit Object(s)" /></td>
<td>Edit &gt; Edit Object(s)</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Move By" /></td>
<td>Draw &gt; Move By</td>
</tr>
</tbody>
</table>
Verification Toolbar

See “The Verification Error Navigator” on page 939 for instructions on using these commands.

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="DRC" /></td>
<td>Tools &gt; DRC</td>
</tr>
<tr>
<td><img src="image" alt="Extract" /></td>
<td>Tools &gt; Extract</td>
</tr>
<tr>
<td><img src="image" alt="DRC Box" /></td>
<td>Tools &gt; DRC Box</td>
</tr>
<tr>
<td><img src="image" alt="DRC Setup" /></td>
<td>Tools &gt; DRC Setup</td>
</tr>
<tr>
<td><img src="image" alt="Extract Setup" /></td>
<td>Tools &gt; Extract Setup</td>
</tr>
<tr>
<td><img src="image" alt="Enable Interactive DRC" /></td>
<td>Tools &gt; Enable Interactive DRC</td>
</tr>
<tr>
<td><img src="image" alt="Interactive DRC" /></td>
<td>Setup &gt; Design &gt; Interactive DRC</td>
</tr>
</tbody>
</table>

Compact Layer Palette

L-Edit provides two layer palettes, both of which display all layers in a design and offer certain layer controls. Either or both palettes can be open at once, and both can float or be docked.

The Compact Layer Palette is a grid of icons that replicate the color and pattern of each layer, to provide a quick way to select a layer when you are drawing. Please refer to “Using the Compact Layer Palette” on page 194 and “Controlling Layer Display” on page 191 for further information.

Base Point Toolbar

See “Base Point Mode” on page 324 for a description of how to use this toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Base Point" /></td>
<td>Use Base Point</td>
</tr>
</tbody>
</table>
### Object Snap Toolbar

See “Snapping Existing Geometry to the Manufacturing Grid” on page 305 for a description of how to use this toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Pick Base Point" /></td>
<td>Pick Base Point</td>
</tr>
<tr>
<td><img src="image" alt="Pick Base Point" /></td>
<td>(untitled) Enter coordinates to locate a base point.</td>
</tr>
</tbody>
</table>

### Node Highlighting Toolbar

See “The Node Highlighting Toolbar” on page 519 for a description of how to use this toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Pick Base Point" /></td>
<td>Extract Connectivity</td>
</tr>
<tr>
<td><img src="image" alt="Pick Base Point" /></td>
<td>Highlight Node</td>
</tr>
<tr>
<td><img src="image" alt="Pick Base Point" /></td>
<td>Highlight by Name</td>
</tr>
<tr>
<td><img src="image" alt="Pick Base Point" /></td>
<td>Zoom to Node</td>
</tr>
<tr>
<td><img src="image" alt="Pick Base Point" /></td>
<td>Toggle Markers</td>
</tr>
<tr>
<td><img src="image" alt="Pick Base Point" /></td>
<td>Clear Markers</td>
</tr>
<tr>
<td><img src="image" alt="Pick Base Point" /></td>
<td>Enable Node Highlighting</td>
</tr>
<tr>
<td><img src="image" alt="Pick Base Point" /></td>
<td>Node Highlight Setup</td>
</tr>
</tbody>
</table>
**Database Toolbar**

L-Edit has a toolbar for certain database operations. See “The Database Toolbar” on page 226 for a description of this toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Toggle reserve of cell view file on disk" /></td>
<td>Toggle reserve of cell view file on disk</td>
</tr>
<tr>
<td><img src="image" alt="Save cell" /></td>
<td>Save cell</td>
</tr>
<tr>
<td><img src="image" alt="Save all" /></td>
<td>Save all</td>
</tr>
<tr>
<td><img src="image" alt="Update cell" /></td>
<td>Update cell</td>
</tr>
<tr>
<td><img src="image" alt="Update all" /></td>
<td>Update all</td>
</tr>
</tbody>
</table>

**Alignment Toolbar**

See “Aligning and Distributing Objects” on page 335 for a description of this toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Left" /></td>
<td>Draw &gt; Align &gt; Left</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Horizontal Center" /></td>
<td>Draw &gt; Align &gt; Horizontal Center</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Right" /></td>
<td>Draw &gt; Align &gt; Right</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Top" /></td>
<td>Draw &gt; Align &gt; Top</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Vertical Center" /></td>
<td>Draw &gt; Align &gt; Vertical Center</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Bottom" /></td>
<td>Draw &gt; Align &gt; Bottom</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Distribute Horizontally" /></td>
<td>Draw &gt; Align &gt; Distribute Horizontally</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Distribute Evenly Horizontally" /></td>
<td>Draw &gt; Align &gt; Distribute Evenly Horizontally</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Distribute Evenly Vertically" /></td>
<td>Draw &gt; Align &gt; Distribute Evenly Vertically</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Tile Horizontally" /></td>
<td>Draw &gt; Align &gt; Tile Horizontally</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Tile Vertically" /></td>
<td>Draw &gt; Align &gt; Tile Vertically</td>
</tr>
<tr>
<td><img src="image" alt="Draw &gt; Align &gt; Tile as a 2D Array" /></td>
<td>Draw &gt; Align &gt; Tile as a 2D Array</td>
</tr>
</tbody>
</table>

**MultiGrid Toolbar**

See “Using the Multigrid Toolbar to Toggle Grid Size” on page 207 for a description of this toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Mouse Grid 1" /></td>
<td>Mouse Grid 1</td>
</tr>
<tr>
<td><img src="image" alt="Mouse Grid 2" /></td>
<td>Mouse Grid 2</td>
</tr>
<tr>
<td><img src="image" alt="Mouse Grid 3" /></td>
<td>Mouse Grid 3</td>
</tr>
<tr>
<td><img src="image" alt="Max Grid" /></td>
<td>Max Grid</td>
</tr>
</tbody>
</table>
Chapter 5: The L-Edit Interface

Elements of the User Interface

**Customizing Toolbars**

You can add buttons for existing commands to existing L-Edit toolbars, add entirely new toolbars, and add new buttons for entirely new commands to either new or existing toolbars.

To customize toolbars, right-click anywhere in the toolbar area and click on **Customize** in the context-sensitive menu.

<table>
<thead>
<tr>
<th>Button</th>
<th>Menu Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Manufacturing Grid]</td>
<td>Manufacturing Grid</td>
</tr>
<tr>
<td>![Make Coarser]</td>
<td>Make Coarser</td>
</tr>
<tr>
<td>![Make Finer]</td>
<td>Make Finer</td>
</tr>
<tr>
<td>![Set Nudge from Snap Grid]</td>
<td>Set Nudge from Snap Grid</td>
</tr>
<tr>
<td>![Mouse Snap Grid]</td>
<td>Mouse Snap Grid</td>
</tr>
</tbody>
</table>

This opens the **Customize** dialog, to the **Toolbars** tab. Note that in this dialog the checkmarks control only whether or not a toolbar is displayed. The buttons apply only to the toolbar that is highlighted, and will be applied even if a toolbar is
not currently displayed.

![Customize dialog box](image)

**Reset** returns an existing toolbar to the default display settings for aspects such as icon size, tooltips, etc.

The **New**, **Rename** and **Delete** functions apply only to custom toolbars.

**Adding a Command to a Toolbar**

Use the **Commands** tab to add a button for an existing command to any toolbar.

[1] Right-click in the toolbar area, select **Customize** and then the **Commands** tab.
Pick the desired command from the **Categories** list (or use All Commands for a complete list of available commands), then simply click-and-drag the command from the right column to the desired toolbar.

L-Edit will insert a button displaying the command text, or an icon if one is already defined.

**Adding a New Toolbar**

1. Right-click in the toolbar area, select **Customize** and click on the **New** button.
2. Enter the desired name in the **New Toolbar** window and click OK to display it in the toolbar area.

Note that although it has a name, the new toolbar is small and blank when first placed – you may have to search a bit to find it.

If you have just added the toolbar, you can click-and-drag from the **Commands** pane to add an existing command. Otherwise, right-click in the toolbar area, select **Customize**, highlight a menu in the **Categories** pane then click-and-drag the desired command from the **Commands** pane to your new toolbar.

**Adding a New Command Button**

There are two important requirements for adding a new command to L-Edit. You must write and execute a TCL function to perform the desired command, and you must create a custom button for the command that has the same name as the TCL function.

As long as the TCL function is loaded into L-Edit during the current editing session, L-Edit will run the function when you press the custom button to execute the operation. Lastly, if you want a button to work in subsequent sessions, you will need to save it to an L-Edit startup folder.

**Example: Adding a Button to Toggle Snap Grid Size**

Please follow the steps in this example to learn how to add a new command button. In this example we will add a toolbar with a new button that draws a rectangle on the current layer.
Enter the following TCL function in the L-Edit command window.

```tcl
proc makeBox {} {rect -llx 0 -lly 0 -urx 100 -ury 100}
```

Create a new toolbar named "myToolbar" (see "Adding a New Toolbar").

If you have just added the toolbar you can click on the Commands tab, scroll down to the bottom of the list and select Custom. If not, right-click in the toolbar area, select Customize, click on the Commands tab, and scroll down to the bottom of the list and select Custom.

Grab the text "Execute button text as Tcl" from the right pane and drag it to the newly created toolbar.

For custom buttons you must replace “Execute button text as Tcl” with the name of the TCL function you want to use. With the “Execute button text as Tcl” button highlighted, right-click and enter the function name “makeBox” in the name field.

You can now press the makeBox button and it will create a box on the current layer.
The makeBox script will execute for the duration of the current session. To execute tcl commands that you create each time L-Edit launches you must save the TCL commands in a file, and source the file when L-Edit is launched.

**Adding a Button using the tcl workspace command**

The workspace command may be used to create a toolbar button that is bound to a tcl command. The `workspace userbutton set` command creates a new toolbar called **User Toolbar**, and creates a toolbar button bound to the specified tcl command. The workspace userbutton set command may be issued multiple times to place multiple buttons on the User Toolbar.

```
workspace userbutton set <commandName>
```

**Docking Views**

Key functional operations in L-Edit are accessed from dockable windows. To show or hide these windows, use **View > Docking Views** or right-click in the toolbar area to launch the checklist shown below.

```
✓ Library and Cell Lists
✓ Layer Palette
✓ Command Line and Log Window
✓ Verification Navigator
✓ SDL Navigator
✓ Aerial View

Hide Docked Views
```

Checking **Hide Docked Views** temporarily hides all of the docking views that are open, to clear a maximum amount of screen space for the layout window. They are restored when “Hide Docked Views” is unchecked.

**Libraries Navigator**

See “The Libraries Navigator” on page 217 for details on using this toolbar.
Layer Palette

L-Edit provides two layer palettes, both of which display all layers in a design and offer certain layer controls. Either or both palettes can be open at once, and both can float or be docked.

The Layer Palette displays a list of layer names and icons that replicate the color and pattern of each layer, to provide a quick way to select a layer when you are drawing and additional features for layer display and manipulation. Please refer to “Using the Layer Palette” on page 196 and “Controlling Layer Display” on page 191 for further information.

and Log Window

L-Edit includes a interface where you can enter basic commands and related parameters so you can perform repeatable, coordinate-specific object manipulations and command scripting. The same window also displays log files for many operations, for confirmation and re-entry as a command. See “Editing” on page 362 for instructions on using this tool.

Verification Navigator

The Verification Navigator displays a scrollable tree of DRC rules and violations for the active cell. After running DRC, you can use the Verification Navigator to step through and display errors in detail. See “The Verification Error Navigator” on page 939 for instructions on using this tool.

SDL Navigator

The schematic-driven layout (SDL) Navigator let you associate a netlist with any layout cell, and provides navigation tools to identify required interconnections. It also can automatically generate layout corresponding to subcircuits and devices. See “Schematic-Driven Layout (SDL)” on page 462 for more details.

Aerial View

The aerial view shows the position of the current viewing window relative to the cell boundary. The location of the viewing window is shown in blue outline, the cell boundary is shown in either black or white outline (depending on the background color of the layout window). You can also use the aerial view to move the position of the viewing window, either by clicking once to keep it the same size but
relocate the center, or with a click and drag movement to change the size, location or both of the viewed area.

![Aerial View](image)

**Status Bars**

There are three status bars associated with L-Edit: the status bar, the mouse button bar, and the locator bar. To show or hide a status bar, use View > Status Bars or the toolbar context-menu.

![Status Bars](image)

**Status Bar**

The status bar, located at the bottom of the L-Edit window, displays context-sensitive information about items in the interface in two separate panes.

| Selection: AA Polygon (Metal 1) Width=5.273, Height=6.000, Area=42.936, Perimeter=28.622, Vertices=1  | Mode: Drawing |

The left pane displays regular L-Edit status as indicated in the following table.

<table>
<thead>
<tr>
<th>Status</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the pointer is in the Compact Layer palette:</td>
<td>The name of the layer the pointer is over. If the layer is generated, the Boolean formula for that layer is also shown.</td>
</tr>
<tr>
<td>When a menu command is highlighted:</td>
<td>A description of the command.</td>
</tr>
<tr>
<td>When a single object is selected:</td>
<td>The type, layer, and size of the object. For cell instances and arrays, the name of the instanced cell.</td>
</tr>
<tr>
<td>When multiple objects are selected:</td>
<td>The count, by type, of the selected items (for example, 4 boxes, 1 circle, 3 ports, and 1 instance).</td>
</tr>
</tbody>
</table>
The right pane shows the current L-Edit mode (for example Nibble or Zoom Box) and provides the Details button ( ) which when pressed creates a text file with a textual description of the selected object, such as **Box, A=1082.25, P=154.0000, W=58.5000, H=18.5000 on Layer 'Poly'**.

The **View > Selection Detail** command is an enhanced version of the status bar Details button. It opens a list in the Command window of all the selected objects, by shape, and details the dimensions (width, height, area and perimeter), layer and lower left location for each shape.

<table>
<thead>
<tr>
<th>Status</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>When a persistent ruler is selected:</td>
<td>The ΔX &amp; ΔY of a ruler or measurement operation.</td>
</tr>
<tr>
<td>When the pointer is in a toolbar:</td>
<td>The function of the pointed-to tool.</td>
</tr>
<tr>
<td>All other times:</td>
<td>Either Ready or blank.</td>
</tr>
</tbody>
</table>

The **View > Display > Cycle Locator Bar Format** (shortcut *Shift+F*) will cycle through the different locator bar display types. See **“Setup Application–General” on page 63** for the other coordinate display modes.
Coordinate System

L-Edit shows object sizes, distances, and positions in any of six physical units, called display units: Microns, Mils, Millimeters, Centimeters, Inches, or a custom unit (if one is defined).

Custom units are defined in the **Setup Design—Technology** dialog. This dialog also lets you set the default value of display units. For its own computation, L-Edit uses internal units (30-bit signed integers). The relation between internal units and physical (technology) units is also defined in the same dialog. For further information, see “Setup Design—Technology” on page 81.

Display units do not affect the physical dimensions of your layout. Rather, they determine the system of units in which L-Edit reports physical lengths, areas, and positions. Display units also determine the units used to specify spatial parameters in various L-Edit dialogs.

For example, if display units are set to microns, \((a, b)\) refers to coordinates microns.

If you change the display units to millimeters, the same coordinates \((a, b)\) will be shown in millimeters:

**Relative Coordinate Display**

When you press Q, the locator changes to relative coordinate display mode. In this mode, the locator displays the coordinates of the pointer’s position relative to its position when Q was pressed. The third number represents the distance between the pointer’s current position and its initial position. When you press Q again, the display goes back to the default mode.

**Layout Area**

The area available for drawing objects is called the layout area. The origin of the coordinate system \((0,0)\) is indicated with a cross-hair marker, which you can hide or display using View > Display > Origin. Optional displays of major and minor grids provide a set of convenient locating points, which you can hide or display using View > Display > Major Grid and View > Display > Minor Grid. You can adjust the spacing of the major and minor grid points using Setup Design—Grid.

Note that, depending on your hardware, it is possible to crash the application by having too many windows open in L-Edit.

**Text Editor**

L-Edit provides a text editor with syntax highlighting for several file types. The File > New command automatically launches the text editor if you have picked any of the following file types:
• Text
• UPI Macro
• SPICE netlist
• Calibre, Dracula and Assura command files

**Standard Navigation Buttons in List Dialogs**

Use the icons below to manage list dialogs such as Setup > Design > Libraries, Valid Vias, etc.

- **Adds** a new file to the list and puts you in edit mode for that file.
- **Deletes** the selected file from the list.
- Moves the selected file **up** one position in the list.
- Moves the selected file **down** one position in the list.

**Environment Variable Usage in File Paths**

File paths in all Tanner applications allow the use of Environment Variables. An environment variable named VARIABLE may be referenced in three ways:

- %VARIABLE%
- $VARIABLE
- ${VARIABLE}
6 Viewing the Layout

At times you may want to display only a portion of your design. Hiding various layout elements can be useful in two ways:

- It speeds up screen redraw by reducing the amount of geometry that must be rendered on screen.
- It allows you to focus more effectively on the particular geometry you are working with.

This chapter explains how to show and hide layout interface elements, layers, objects, and instance contents. It also explains how to view different levels of hierarchy and zoom and pan in the design. All commands apply to all cells in the active file.

Refreshing the Screen

Use View > Redraw or press the Space bar on the keyboard to refresh the screen. L-Edit redraws the layout in the current view.

Displaying Layout Interface Elements

Use View > Display to turn the display on and off for each layout interface element. A check mark next to the item indicates the item is on (displayed).

<table>
<thead>
<tr>
<th>Insides</th>
<th>Display</th>
<th>Hierarchy Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goto...</td>
<td>Home</td>
<td>Home Exchange</td>
</tr>
<tr>
<td>Zoom</td>
<td>Pan</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objects</th>
<th>Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Navigator</td>
<td></td>
</tr>
<tr>
<td>Toolbars...</td>
<td></td>
</tr>
<tr>
<td>Status Bars...</td>
<td></td>
</tr>
<tr>
<td>Docking Views</td>
<td></td>
</tr>
<tr>
<td>Redraw</td>
<td>Space</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Icons</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Arraneger Design Navigator</td>
<td></td>
</tr>
<tr>
<td>Show All</td>
<td>Hide All</td>
</tr>
</tbody>
</table>

Turns the visibility of lower-level geometry on and off when View > Insides > Toggle Insides is off. (See “Displaying Instance Insides” on page 202.) When Icon is on, L-Edit displays objects that reside on the Icon layer within an instance but hides the rest of the instance’s contents.
Many IC fabricators allow the identification of an Icon layer whose geometry is ignored during fabrication. To specify exactly one L-Edit layer as the Icon layer, use Setup > Special Layers. Objects on the Icon layer can be used to annotate an instanced cell or highlight one cell’s relationship to another.

**Arrays**

Turns the visibility on and off for instance contents in arrays in the active cell. When on, all arrays are shown in full, with all repeated instances visible. When off, each array is displayed as an outline in which only the first instance is rendered.

**Ports**

Turns the visibility on and off for ports within the first level of hierarchy for all displayed instances. This command does not affect top-level ports, which are always shown, or ports on hidden layers, which are never shown.

**Major Grid**

Turns the visibility of the major grid display on and off when the Grid layer is shown.

(For information on showing and hiding layers, see “Hiding Layers” on page 192.)

The grid may not be visible at all levels of magnifications.

**Minor Grid**

Toggles visibility of the minor grid display.

**Origin**

Turns on and off the visibility of the crosshair marker that indicates the origin (0,0) when the Origin layer is shown.

**Mouse Hints**

Turns on and off the visibility of mouse tooltips displayed beneath the mouse pointer in layout windows. Mouse hints show the operation associated with each mouse button as vertical text, icons, or both.

**Cycle Locator Bar Format (Shift+F)**

Cycle through the different locator bar display types.

**Toggle Markers**

Toggles marker display on and off.

**Clear Markers**

Deletes all markers from the active file.

### Showing and Hiding Objects

You can show or hide all objects of a specific type in a design. When you hide objects, L-Edit indicates the hidden state by shading the object icon on the Drawing toolbar.

While objects are hidden, you cannot draw, select, edit, move, or delete them.
Choose **View > Objects** to display the following menu:

<table>
<thead>
<tr>
<th>Objects</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A check by an object type indicates the object is currently visible in the design. To make all objects visible, select **Show All**. To hide all objects in the design, select **Hide All**.

L-Edit treats pie wedges and tori as polygons when showing and hiding objects.

You can also show and hide objects with a right-click over the Drawing toolbar, which opens the menu shown below.

This menu is context-sensitive—**Show** pertains to the specific object icon the pointer is over at the time you activate the menu. A check mark next to **Show** indicates that objects of that type are currently visible in the design; no check mark indicates that objects of that type are hidden. To make all objects
visible, select **Show All**. To hide all objects except for a particular object type, position the pointer directly over the icon for the desired object and select **Hide All**.

You can also use a middle-click on the icons on the Drawing toolbar to toggle the hidden state of an object type.

You can also use a middle-click on the icons on the Drawing toolbar to toggle the hidden state of an object type.

To hide all objects except for one object type, position the pointer directly over the desired object icon in the Drawing toolbar and click the H/S ALL (**Ctrl**+HIDE/SHOW) mouse button.

---

**Controlling Layer Display**

L-Edit provides two layer *palettes*, the **Layer Palette** and the **Compact Layer Palette**. You can have either or both palettes open at once.

**Sorting Layers**

Both palettes can be filtered using a pull-down category list to display just those layers that are drawn, in use, generated or special. Category or filter constraints that you enter in the Layer Palettes are applied to all layer lists and layer drop-down menus in L-Edit.
You can also select **Filter**, and then enter text in the entry field to limit the palette display to layers that include exactly the characters entered anywhere in their name.

In setup layers and in dialogs such as Edit Objects, you can enter characters to sort the layer list, with an option to sort by Layer Name or Purpose. L-Edit will also suggest layer names as you type.

There is a set of UPI functions that let you add newly created layers to a custom Layer Palette filter, switch between layer palettes and quickly name or rename them.

**Hiding Layers**

You can show or hide all objects on a specific layer in a design. When a layer is *hidden* you cannot view, draw, select, edit, move, or delete objects it contains.

To set which layers are shown or hidden, you can choose **View > Layers**, middle-click a layer icon on either of the layer palettes, right-click to open the context menu or use the **Hidden** checkbox in **Setup Layers**.

**Show/Hide Layers Using the View > Layers Command**

When you select **View > Layers**, L-Edit displays the following menu:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show [Layer name]</strong></td>
<td>Refers to the active layer. A check mark next to <strong>Show [Layer name]</strong> indicates that objects on that layer are currently visible in the design; no check mark indicates the objects on that layer are hidden.</td>
</tr>
<tr>
<td><strong>Show All</strong></td>
<td>Makes objects on all layers visible</td>
</tr>
<tr>
<td><strong>Hide All</strong></td>
<td>Hides objects on all layers except those on the active layer</td>
</tr>
<tr>
<td><strong>Show Generated</strong></td>
<td>Shows objects on all generated layers.</td>
</tr>
<tr>
<td><strong>Hide Generated</strong></td>
<td>Hides objects on all generated layers except the layer indicated by the pointer if the indicated layer is a generated layer.</td>
</tr>
</tbody>
</table>

**Show/Hide Layers Using the Mouse**

To toggle whether an individual layer is shown or hidden from either layer palette, position the pointer directly over the desired layer icon and click the HIDE/SHOW (middle) mouse button. L-Edit will toggle the layer state to hidden if it is shown, or to showing if it is hidden.

You can use **Ctrl+middle click** to HIDE ALL or SHOW ALL for all but one layer (since at least one layer must be visible), the layer over which the pointer is positioned when you perform the operation.

L-Edit indicates the hidden state differently on each of the layer palettes. The indicator for the hidden state in the **Compact Layer Palette** is diagonal hatching on an icon, as shown below.

All layers showing.  
Middle layer hidden.
The indicator for the hidden state in the **Layer Palette** is gray highlighting over the name and a check in the hidden checkbox, as shown below.

![Layer Palette](image)

**Locking Layers**

You can lock all objects on a specific layer in a design. When a layer is *locked*, you cannot draw, move, or edit objects on that layer.

To lock or unlock a layer, you can click in the checkbox in the lock column, right-click a layer icon on the one of the layer palettes to open a context-sensitive menu or check the **Hidden** option in **Setup Layers**.

In the Compact Layer Palette context menu, the check next to **Lock PNP Emitter ID** indicates that layer is currently locked.

L-Edit indicates the locked state differently on each of the layer palettes. The locked indicator in the Compact Layer Palette is grid hatching as well as a check in the box.

Grid hatching indicates a locked layer, as in the two leftmost icons.
The locked indicator in the Layer Palette is grid red text and a shaded background as well as a check in the box.

Red text indicates a locked layer. In this example, layer background is locked.

Protecting Layers

You can protect a specific layer so that you cannot select any objects it contains. There is no visual indication of the protected state.

To protect a layer, you can click the box in the protected column of either layer palette, choose View > Layers, middle-click a layer icon on either of the layer palettes, right-click to open the context menu or use the Protect checkbox in Setup Layers.

Using the Compact Layer Palette

The Compact Layer Palette is a grid of icons that replicate the color and pattern of each layer to provide a quick way to select a layer when you are drawing.

To display the Compact Layer Palette, use View > Toolbars and click on the Compact Layer Palette checkbox.
A right-click on any layer icon in the palette opens a menu with the options to show, hide or lock layers; open the **Setup Layers** dialog, and change the size of the icons, as shown below.
Using the Layer Palette

The default display of the Layer Palette is tabbed together with the Libraries navigator. You use the **Layer Palette** to select a layer for drawing operations, to manage layer settings and to manage layer display.

(Currently selected layer) This area indicates the currently selected layer. A right-click here opens a context-sensitive menu (see “Right-Click Menu in the Layer Palette” on page 199). A double-click here opens Setup Layers for the layer indicated.
(Category pull-down menu) Use this list to limit the palette display to just one category of layer purpose pairs:

- **All** shows all layer types, including those that are not valid.
- **Filter** activates the filter field so you can limit the layer list by name. Note that category or filtering constraints entered here are applied to all layer lists in L-Edit.
- **In Use in File** shows only the layers used in the active file.
- **In Use in Cell** only shows layers used in top-level geometry of a cell and does not show layers used by instances within the cell.
- **In Use in Cell + Hierarchy** shows layers used in top-level geometry and the layers used in instances contained in the cell. This is helpful, for example, when a cell contains just instances and no primitive geometry.
- **Generated** shows only generated layers.
- **Special** shows only special layers.

(Text-entry filter field) When the Filter category is selected, enter text to list only those layers containing the characters entered in this field any place in their name. For example, “la” in the filter field will display layers “label,” “blade” and any others with “la” in their name.

(Layer Palette header) The checkboxes in these columns enable the locked, hidden or protected state of a layer. See “Buttons in the Layer Palette Header,” below.

(Buttons for instance and port selectability) Use these buttons to toggle selectability for the active design.

- Indicates that instances can be selected.
- Indicates that instances cannot be selected
- Indicates that ports can be selected.
- Indicates that ports cannot be selected

(Name, if any, of the palette in use) This field displays the name of the palette that is currently loaded. **<All layers>** is the default.

It is also a pull-down menu you can use to select previously saved palettes, or to create and save palettes using the **<Setup>** option (see “Creating and Saving a Layer Palette” on page 200).
Buttons in the Layer Palette Header

You can click on a button in the header to sort the list by that column value. The columns may be scaled by clicking and dragging the separator bars. A check in the appropriate box enables the locked, hidden or protected state of a layer.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Purpose number</th>
<th>Locked</th>
<th>Hidden</th>
<th>Protected</th>
<th>Layer number</th>
<th>data type</th>
<th>GDSII number</th>
</tr>
</thead>
<tbody>
<tr>
<td>This column displays the layer name.</td>
<td>This column displays the layer purpose number.</td>
<td>When a layer is locked, geometry cannot be drawn, moved, or edited. The layer name appears with grey shading in the layer palette.</td>
<td>When a layer is hidden it is not displayed in layout. The layer name appears in red text in the layer palette.</td>
<td>When a layer is protected, objects on the layer cannot be selected. The layer name appears in blue text in the layer palette.</td>
<td>This column displays the layer number.</td>
<td>This column displays the data type.</td>
<td>This column displays the GDSII layer number.</td>
</tr>
</tbody>
</table>

Right-Click Menu in the Layer Palette Header

A right-click on any of the header buttons opens this menu:

- Ascending
- Descending
- Unsorted
- Setup Layers...
- Setup Layer Palettes...
- Options
Ascending
Descending
Unsorted
Setup Layers
Setup Layer Palettes
Options

Right-Click Menu in the Layer Palette

A right-click on any layer name in the Layer Palette opens the following shortcut menu.

Show
Hide
Protect
Unprotect
Lock
Unlock
Setup Layers...
Setup Layer Palettes...

Options

Show

Hide

Protect

Show [layer name]
Show all
Show generated

Show
Hide
Protect

Show [layer name]

Show all

Show generated

Options

Show Header
Unsorted

Show Purpose column
Show Purpose Number column
Show Locked column
Show Hidden column
Show Protected column
Show Layer Number column
Show GIS data type (CR) column
Show GIS x column
Increase Icon Size
Decrease Icon Size

Show
Check one of these options to set which layers will be displayed—the highlighted layer, all layers or just generated layers.

The layer in Show [layer name] at the top of the menu corresponds to the icon the pointer was over when the menu was activated. To make objects on all layers visible, select Show All.

Hide
Check this option to set which layers will be hidden—the highlighted layer, all layers or just generated layers.

To hide objects on all layers except those on a particular layer, position the pointer directly over the desired layer icon and select Hide All.

Protect
Check this option to protect the highlighted layer or all layers.
Unprotect
Check this option to unprotect the highlighted layer or all layers.

Lock
Check this option to lock the highlighted layer or all layers.

Unlock
Check this option to unlock the highlighted layer or all layers.

Setup Layers...
Click this option to open the Setup Layers dialog (see “Layer Setup” on page 89).

Setup Layer Palettes...
Click this option to open the Setup Layer Palettes dialog (see “Creating and Saving a Layer Palette,” below).

Options
Opens a checklist where you can control whether the header is shown, whether the layer icons are sorted, which columns to display, and the size of layer icons, as shown below.

- Show Header
- Unsorted
- Show Purpose column
- Show Purpose Number column
- Show Locked column
- Show Hidden column
- Show Protected column
- Show Layer Number column
- Show GDS data type (DT) column
- Show GDS # column

Increase Icon Size
Decrease Icon Size

Creating and Saving a Layer Palette

Setup Layer Palettes lets you customize, name and save and retrieve different layer palettes. The layers listed on the right side will be included in a layer palette; those on the left side list will not be.
Use a right-click in the Layer Palette list or **Setup > Setup Layer Palettes** to open this dialog. When you first open it, the dialog will be blank. When you press **New**, the right side layer list will be populated with the layers defined in the active design, and you will prompted to name your new layer palette.

Highlight layers in the Layer palette list using the Shift and Ctrl buttons for multiple selection. You can then use the Exclude button to remove a layer from the layer palette display and the Include button to include a layer in the display. You can also click to place a check in the appropriate box to lock, hide or protect a layer. **Rename** lets you rename an existing Layer palette.

**Note:** **Delete** is effective immediately. You will not have the option to confirm deletion of a named layer palette.

Once you have defined a non-default layer palette, you can use the pull-down menu at the bottom of the Layer Palette tab to select from the saved layer palettes, to relist all defined layers in the design, or to open the Setup Layer Palettes dialog.

The **Reorder** button will invoke a dialog that can be used to reorder the appearance of the palettes as they appear in the pull-down menu at the bottom of the Layer Palette.

### Viewing Layout Hierarchy

To clarify different parts of your design, you can show and hide different levels of hierarchy. Select one of the following commands:

- **View > Hierarchy Level > Show one more level**
- **View > Hierarchy Level > Show one less level**

To display a specific number of levels, select **View > Hierarchy Level > View hierarchy level**. L-Edit displays the **View Hierarchy Levels** dialog.

The view level is relative to the top level of the cell. If you specify zero or a number higher than the number of levels that exist in the design, L-Edit displays all hierarchy levels.
You can also select the number of levels to show from the drop-down list in the Standard toolbar, illustrated below. However you set the view hierarchy, L-Edit saves the information in the TDB file.

![View Hierarchy Levels Dialog](image)

Click the menu in the Standard toolbar to specify how many levels of hierarchy you want to display. Selecting Other opens the View Hierarchy Levels dialog.

You can also use the The Design Navigator (page 242) to display a hierarchical list of all cells in the design, including information such as whether the cell is locked, which instances it contains, its DRC status, etc.

Use View > Arrange Design Navigator (shortcut F10) to quickly arrange all open windows with the Design Navigator to one side and the layout windows, each the same size, on the other.

### Displaying Instance Insides

There are two ways to display instances in your design:

- Completely, so that you can see all of the objects (the insides) in the instanced cells
- As outlines with just the name of the instanced cell. Displaying instances as outlines decreases the amount of time L-Edit takes to redraw the screen and can help clarify different portions of the design.

There is a set of View commands that control the visibility of insides in a design for either all instances, only selected instances, or only leaf-level cells. (A leaf-level cell contains no instanced cells.)

The following table describes these commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View &gt; Insides &gt; Toggle Insides</td>
<td>Ctrl+I or Tab</td>
<td>Shows or hides the insides of all instances at all levels of the hierarchy. When insides are hidden, ports one level down in the hierarchy remain visible.</td>
</tr>
<tr>
<td>View &gt; Insides &gt; Show Cell Insides</td>
<td>S</td>
<td>Shows the insides of the selected instance(s).</td>
</tr>
<tr>
<td>View &gt; Insides &gt; Hide Cell Insides</td>
<td>D</td>
<td>Hides the insides of the selected instance(s).</td>
</tr>
<tr>
<td>View &gt; Insides &gt; Show Leaves</td>
<td>Alt+B</td>
<td>Shows the insides of all leaf-level cells in the design.</td>
</tr>
<tr>
<td>View &gt; Insides &gt; Hide Leaves</td>
<td>Alt+L</td>
<td>Hides the insides of all leaf-level cells in the design.</td>
</tr>
</tbody>
</table>
You can also use “Setup Application–Rendering” (page 72) to set a default pixel size below which instance insides will be hidden.

**Displaying Instance Insides While Drawing and Editing**

During drawing operations, the Tab key toggles display of the object being drawn from fill mode, where objects are rendered completely filled, to outline mode, where objects are rendering with their insides hidden.

To set outline mode as the default during drawing, uncheck the Fill objects when editing/drawing checkbox in Setup > Application—Rendering.

After the drawing operation is completed the drawing mode will revert to the default setting.

Similarly, when an instance is being moved, the Tab key acts as a three state button that cycles through the following display levels:

- outline only
- outline of the instance and outlines of all first level objects inside the instance
- outline of the instance and fills of all first level objects inside the instance, with the exception of other instances.

**Zooming and Panning**

You can zoom into or out of the current view to see more or less of the design. You can also move (pan) around the design to see different portions of it at the current level of magnification. All commands affect only the active cell.

**Zooming**

There are four L-Edit commands related to zooming. All of them change the magnification of the current view and do not affect the position and location of objects in the design. Increasing the magnification (zooming in) causes the objects to look larger; decreasing the magnification (zooming out) causes the objects to look smaller.

<table>
<thead>
<tr>
<th>Command</th>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View &gt; Home</td>
<td>Home</td>
<td>Changes the magnification so that the view includes all objects in the cell.</td>
</tr>
<tr>
<td>View &gt; Zoom &gt; In</td>
<td>+</td>
<td>Magnifies the view by a factor of two.</td>
</tr>
<tr>
<td>View &gt; Zoom &gt; Out</td>
<td>–</td>
<td>Reduces the magnification of the view by a factor of two.</td>
</tr>
<tr>
<td>View &gt; Zoom &gt; Zoom By...</td>
<td>Alt+Z</td>
<td>Zooms in/out by a specific amount. The zoom amount is relative zoom factor change. Values less than 1, zoom in, while values greater than 1, zoom out. For example, 2 will zoom out by 2x while 0.5 will zoom in by 2x.</td>
</tr>
</tbody>
</table>
Chapter 6: Viewing the Layout

Zooming While Editing In-Place

When you are editing an instance in-place you can zoom to the home view of the cell you are currently editing or the home view of the top cell in the instance hierarchy. To zoom to the home view of the cell you are currently editing, use View > Home or press the Home key. To zoom to the home view of the top cell in the instance hierarchy use Edit > Edit In-Place > View Top Cell or press the End key.

For more information on editing in-place, see “Editing an Original Instance In-Place” on page 253.

Panning

There are eight L-Edit commands related to panning. All of them change the current view and do not affect the position and location of objects in the design.

<table>
<thead>
<tr>
<th>Command</th>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View &gt; Zoom &gt; To Selections</td>
<td>W</td>
<td>Changes the magnification so that the view includes only selected objects in the cell.</td>
</tr>
<tr>
<td>View &gt; Pan &gt; To Selections</td>
<td>Y</td>
<td>Centers the view over the selected objects. Depending on the magnification, all selected objects may not be visible in the resulting view.</td>
</tr>
<tr>
<td>View &gt; Pan &gt; Left</td>
<td>←</td>
<td>Moves the view to the left by one-quarter of its width.</td>
</tr>
<tr>
<td>View &gt; Pan &gt; Right</td>
<td>→</td>
<td>Moves the view to the right by one-quarter of its width.</td>
</tr>
<tr>
<td>View &gt; Pan &gt; Up</td>
<td>↑</td>
<td>Moves the view up by one-quarter of its height.</td>
</tr>
<tr>
<td>View &gt; Pan &gt; Down</td>
<td>↓</td>
<td>Moves the view down by one quarter of its height.</td>
</tr>
<tr>
<td>View &gt; Pan &gt; Pan By...</td>
<td>Alt+F</td>
<td>Pan by DX DY, where 0 &gt; Right or Up, &lt; 0 Left or Down, using a space, comma or tab to delimit your entry.</td>
</tr>
<tr>
<td>View &gt; Pan &gt; To Cell Edge</td>
<td></td>
<td>Shifts the view so that the edge of the view is flush with the edge of the contents of the cell. See “Pan To Cell Edge,” below.</td>
</tr>
<tr>
<td>View &gt; Pan &gt; Object Pan &gt; Left</td>
<td>J</td>
<td>Pans to the left side of an object. For wires, it pans to the leftmost endpoint. For boxes and polygons, it pans to the center of the left size of the object’s minimum bounding box (MBB).</td>
</tr>
</tbody>
</table>
**Pan To Cell Edge**

*View > Pan > To Cell Edge* shifts the view so that the edge of the view is flush with the edge of the contents of the cell.

<table>
<thead>
<tr>
<th>Command</th>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View &gt; Pan &gt; Object</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pan &gt; Right</td>
<td>K</td>
<td>Pans to the right side of an object. For wires, it pans to the rightmost endpoint. For boxes and polygons, it pans to the center of the right size of the object’s minimum bounding box (MBB).</td>
</tr>
<tr>
<td>View &gt; Pan &gt; Object</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pan &gt; Settings</td>
<td></td>
<td>Controls the view when the endpoint of an object (“object pan”) is displayed. See Object Pan Settings (page 206).</td>
</tr>
</tbody>
</table>

**Pan To Cell Edge**

*View > Pan > To Cell Edge* shifts the view so that the edge of the view is flush with the edge of the contents of the cell.

<table>
<thead>
<tr>
<th>Command</th>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Shift + ←</td>
<td>Shifts the view so that the left edge of the view is flush with the left edge of the contents of the cell.</td>
</tr>
<tr>
<td>Right</td>
<td>Shift + →</td>
<td>Shifts the view so that the right edge of the view is flush with the right edge of the contents of the cell.</td>
</tr>
<tr>
<td>Up</td>
<td>Shift + ↑</td>
<td>Shifts the view so that the top edge of the view is flush with the top edge of the contents of the cell.</td>
</tr>
<tr>
<td>Down</td>
<td>Shift + ↓</td>
<td>Shifts the view so that the bottom edge of the view is flush with the bottom edge of the contents of the cell.</td>
</tr>
</tbody>
</table>
Object Pan Settings

Center view on endpoint

Centers the display on the object’s endpoint, without changing the zoom level.

Center view on endpoint and zoom

Centers the display on the object’s endpoint and zooms in or out until the object’s endpoint and a specified margin (see below) fill the active window.

Margin around endpoint

Sets the margin, in display units, to be displayed around the object’s endpoint when automatic zoom is active.

Zooming and Panning with the Mouse

Use View > Zoom > Mouse, click the mouse zoom button ( ), or press Z to change the functions of the mouse buttons for a single operation. The three buttons become ZOOM BOX (left), PAN (middle), and ZOOM OUT (right). The following table describes each button function.

<table>
<thead>
<tr>
<th>Button</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOOM BOX</td>
<td>Click at a single point to magnify the area around the pointer by a factor of two.</td>
</tr>
<tr>
<td></td>
<td>Click and drag the pointer to specify a rectangular area to which the view will be zoomed. The height-to-width ratio is maintained as closely as possible to the original view.</td>
</tr>
<tr>
<td>PAN</td>
<td>Click to pan the view so that the new center is located at the pointer’s position.</td>
</tr>
<tr>
<td></td>
<td>Click and drag the pointer to pan the view in the direction and through the distance of the pointer’s motion (when the button is released).</td>
</tr>
<tr>
<td>ZOOM OUT</td>
<td>Click to zoom the display window out from the location of the pointer.</td>
</tr>
</tbody>
</table>

After a mouse-controlled viewing operation, the mouse buttons revert to their previous functions.

Mouse Wheel Functions

The mouse wheel works to scroll up and down through any layout, cross section, text or navigator window in L-Edit if no keys are used.
In a layout window, **CTRL + mouse wheel** zooms the window in or out with the cursor location as the center, and **SHIFT + mouse wheel** scrolls the window left or right.

In a cross section window, the mouse wheel will step up or down through process layers.

In a non-window area (title bars, toolbars, etc.) the mouse wheel will scroll through all open document windows and **CTRL + mouse wheel** will scroll through only the windows in the active file.

**Auto-Panning**

*Auto-panning* involves using the mouse to continually pan the view. To activate auto-panning, use **Setup > Application—General** and check the **Auto-panning** option. With auto-panning on, when you draw on object or a selection marquee to the edge of the current view the view automatically pans beyond the edge.

**Moving to Specific Coordinates**

Use **View > Goto** to center the view on specific coordinates in the layout.

The coordinates are in display units, separated by a space or a comma, and are relative to the origin. Coordinates can be typed or copy-pasted into the field.

**Toggling Views**

Use **View > Exchange** or press **X** to return to the previous view after you execute any zoom or pan command. You can use this command to go back and forth between two views.

**Using the Multigrid Toolbar to Toggle Grid Size**

The Multigrid toolbar provides a convenient way to change the current mouse snap grid and the nudge distance, and also to switch between several predefined grid sizes. It is particularly helpful when you have a large change in zoom level.

You can enable or disable this toolbar using the **View > Toolbars** menu item, or by right-clicking in the toolbar area of the application.
The first five buttons from the left set the current mouse snap grid. These buttons are modal—only one can be active at a time.

<table>
<thead>
<tr>
<th>Function</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse Grid 1, 2 &amp; 3</td>
<td><img src="image" alt="Mouse Grid Icons" /></td>
<td>These buttons let you set three predefined mouse snap grid values. Simply click on one of the buttons and enter a value in Mouse Snap Grid field.</td>
</tr>
<tr>
<td>Max grid</td>
<td><img src="image" alt="Max Grid Icon" /></td>
<td>Sets the mouse snap grid to be one internal unit.</td>
</tr>
<tr>
<td>Manufacturing grid</td>
<td><img src="image" alt="Manufacturing Grid Icon" /></td>
<td>Sets the mouse snap grid to the same value as the manufacturing grid.</td>
</tr>
<tr>
<td>Make Coarser</td>
<td><img src="image" alt="Make Coarser Icon" /></td>
<td>Increments the value in the Mouse Snap Grid field upwards or downwards respectively to make the grid coarser or finer, in steps of 1, 2, 5, and 10 display units. This command is not available if the grid is set to “finest” or “manufacturing grid”.</td>
</tr>
<tr>
<td>Make Finer</td>
<td><img src="image" alt="Make Finer Icon" /></td>
<td>Set Nudge From Snap Grid</td>
</tr>
<tr>
<td>Set Nudge From Snap Grid</td>
<td><img src="image" alt="Set Nudge From Snap Grid Icon" /></td>
<td>Sets the nudge amount to the current Mouse Snap Grid value.</td>
</tr>
<tr>
<td>Mouse Snap Grid</td>
<td><img src="image" alt="Mouse Snap Grid Icon" /></td>
<td>Use this numeric field to enter the mouse snap grid values for the buttons 1, 2 &amp; 3, or, when the Set Nudge From Snap Grid button is pressed, the nudge distance. Two space separated values may be entered if different X and Y grid values are desired. Four space separated values may also be entered, with the first two values representing the X and Y grid spacing, and the third and fourth values representing the X and Y offset. Three values may also be entered, with the first two representing the X and Y grid spacing, and the third representing both the X and Y offset.</td>
</tr>
</tbody>
</table>

**Schematic Probing and Selection Synchronization**

L-Edit Probe to Schematic and Synchronize Schematic Selection features can be used to select an instances, net, or ports in layout, and then easily find and select the corresponding object in schematic.

Pressing the Probe button on the Synchronization Toolbar puts L-Edit into schematic probing mode. Pressing the Synchronize button on the Synchronization Toolbar puts L-Edit into schematic synchronization mode.

Using the probe tool you can click on an instance or net in layout, and the instance or net with the same name will be highlighted in schematic, and S-Edit will pop to the front. With Synchronize Selection,
selecting an instance in net in layout using the normal selection methods will select the instance or net with the same name in schematic, but S-Edit will not pop to the front. Probing and Synchronization requires that the Layout and and corresponding Schematic cell names be the same for the particular layout cell being probed.

**Cross-Section Viewer**

L-Edit cross-section views depict the vertical structure of an integrated circuit.

Though actual chips have properties and process artifacts such as smooth height transitions, bird’s beak, and planarization which L-Edit does not model, the cross-section viewer provides a highly readable image of the fabrication process.

Cross-section views are generated from layout by simulating a set of process (fabrication) steps and building the diagram from the substrate up, one layer at a time. These simplified process steps correspond only roughly to the process steps used by the fabricator to create the chip. The *process definition* is derived from a separate text file (see “Process Definition Files” on page 213).

The cross-section viewer simulates three types of process steps:

- **Grow/deposit** to generate new material
- **Etch** to remove material
- **Implant/diffuse** to modify the material nearest the surface

**Grow/Deposit**

New material is generated uniformly in a grow/deposit step. The substance specified in the process step statement is grown or deposited vertically to the specified depth (measured in technology units) on all upward-facing surfaces. The following figure depicts new material deposited in a grow/deposit step.

Oxide growth and metal deposition are both simulated with this type of step. (In reality these fabrication layers and the substrate are manufactured with completely different procedures, but for cross-section viewing purposes, the results can be modeled in the same way.)

**Etch**

An etch step removes material from all areas covered by the specified mask layer. The etch process model involves up to three parameters: the depth, the undercut offset, and the angle. Depths and offsets are measured in technology units; angles are measured in degrees. A cross-section surface resulting
from an etch step with depth $d$, undercut offset $u$, and angle $a$, between points $x_1$ and $x_2$, is shown in the following figure.

Typically, many of the layers to be etched will not be simple drawn mask layers, but will result from logical operations such as AND, OR, and NOT combining several mask layers. Unlike a physical etch that may remove some materials but not others, the simplified etch step removes all materials uniformly. Although nonphysical, the simplified etch captures the important details of most semiconductor fabrication processes.

**Implant/Diffuse**

To simulate the ion-implantation or high-temperature diffusion process that modifies the type of semiconductor nearest the surface, an implant/diffuse step causes the color of the specified mask layer to replace the existing ones from the top surface down to the specified depth in all areas covered by the layer. The implant/diffuse process model involves the same parameters as in the etch model, except that the underlying material is replaced rather than removed.

Again, the mask layer may be a logically derived one. For example, the self-aligned polysilicon gate structure requires a combination of the polysilicon and active mask layers to determine where to show the implanted active, which may be blocked by either field oxide (the NOT of active) or by polysilicon. Operations on layers are specified with the **Setup Layers** dialog. The derived layers are specified for cross-section views the same way as for DRC and extraction layers.

**Operation**

A process definition file must exist prior to generating a cross-section. The layer names in this file must exactly match the layer names in the layout that you wish to view in cross-section. Sample process definition files are provided in the *tech* directory, which is located in the default L-Edit install directory. The process definition files are described in the file *Index.txt*, which is located in the same directory. For information on the syntax of a process definition file, see “Process Definition Files” on page 213.

The cell for which you wish to generate the cross-section must be open. Arrange the display such that a small region of interest (usually a few transistors) is centered in the upper portion of the layout view.
When you choose **Tools > Cross-Section** or press the cross-section button ( ), L-Edit displays the following dialog:

**Generate Cross-Section**

- **Process definition file**
  - Type in the name of the process definition file, or use the **Browse** button to select the file.

- **Pause after first step**
  - Pauses cross-section generation after the first step in the process. To resume cross-section generation, click the **Next Step** button ( ) in the cross-section window.

- **Vertical coordinate (Y)**
  - Sets the vertical coordinate along which the cross-section is generated.

- **Pick**
  - Allows you to set the vertical coordinate graphically. The cursor becomes a horizontal line that can be dragged up or down in the layout. Clicking the mouse button over the desired position reopens the **Generate Cross-Section** dialog, with the graphically selected y-coordinate in the **Vertical coordinate (Y)** field.

- **Exaggeration factor**
  - Sets the magnification factor for the cross-section along the z-axis in terms of a ratio. Since process depths are measured in technology units, the displayed thicknesses of layers in cross-section scale with the current layout magnification. At very large or very small magnifications, it may be impossible to display cross-section views effectively at a 1:1 horizontal-to-vertical aspect ratio. The two fields (numerator and denominator) specify the ratio by which to compress or expand the vertical axis of the cross-section.

- **Auto-fit in window**
  - Sets magnification along the z-axis for maximum visibility.
Display

L-Edit displays the cross-section view in the lower portion of the application interface.

Where the active cell displays a cross-section view:

- You cannot pan, zoom, or edit in the cross-section window.
- You cannot perform an editing operation in other windows associated with the file.
- You cannot resize the layout window.

The split line separating layout from the generated cross-section can be dragged into another location. Double-clicking on this line removes the cross-section view.

To continue normal layout editing, close the cross-section window.

**Single-Step Display**

You can step through the cross-section view one process step at a time. To do click the appropriate button in the cross-section window:

![Button Images]

- First process step
- Previous process step
- Next process step
- Final process step
A tooltip identifying the associated process step appears over these buttons when you point at them. The current step is displayed in the status bar.

Single-step mode is useful for learning the steps involved in fabrication. For instruction in real fabrication processing, a much more detailed process definition could be used.

Single-stepping through a fabrication cross-section that includes all the photoresist and other intermediate processing steps would better communicate the full complexity of today’s fabrication processes. For designers who only want to view final cross-sections, simpler process definitions (such as the example in this chapter) are sufficient and easier to maintain.

**Process Definition Files**

The cross-section process definition file (XST) contains a list of comment statements and process statements. Comment statements begin with a pound sign (#) and continue to the end of the line.

Process statements have the following format:

```
step layer depth label [angle [offset]] [comment]
```

Each process statement begins with a **step** type, one of the following:

- **gd** or grow/deposit
- **e** or etch
- **id** or implant/diffuse

**Layer** is the name of the involved layer. The name of the layer must match the layer name used in the L-Edit TDB file. If the layer name begins with a digit or contains spaces, then the entire name must be enclosed in double-quotes ("…"). The layer name describes something different for each type of step:

- For grow/deposit steps: the layer to be grown/deposited
- For etch steps: the layer to be etched away
- For implant/diffuse steps: the layer to be diffused

A dash (-) in place of a layer name indicates that the process step has no associated rendering information.

**Depth** is a (non-negative) value indicating the depth, measured in technology units. The depth also means different things for different steps:

- For grow/deposit steps: the number of units to grow upward
- For etch and implant/diffuse steps: the number of units downward to apply the step

**Label** is optional. The label may be any string. If it contains spaces, the entire label must be enclosed in double-quotes ("…"). A dash (-) may be used in place of a label.

If desired, two parameters that apply only to etch and implant/diffuse steps are inserted next:

- Etch-implant **angle** (integer)
- Undercut **offset** (non-negative floating-point or integer)

Angles are measured in degrees and must be between 0 and 180; offsets are measured in technology units. The default values are **angle = 80** and **offset = 0**.
Last is an optional comment. The comment begins with a pound sign (#) and continues to the end of the line.

Example

A sample definition for an $n$-well, double-poly, double-metal CMOS process is shown below. Each line (after the header) corresponds to one process step.

File: mORBn20.xst
For: Cross-section process definition file
Vendor: MOSIS: Orbit Semiconductor
Technology: 2.0U N-Well (Lambda - 1.0um, Technology - SCNA)
Technology Setup File: mORBn20.tdb
Copyright (c) 1991-93
Tanner Research, Inc. All rights reserved

L-Edit
Step Layer Name Depth Label [Angle[offset]] Comment
------------- -------------------------- -----------------------------------
gd 10 p- "Well X"3 n- 1. Substrate
id ActPSelNotPoly0.9p+ 75 0 3. p-Implant
id ActNSelNotPoly0.9n+ 75 0 4. n-Implant
id CCD&Act 0.4 - 5. CCD Implant
id "P Base"2 - 6. NPN Base Implant
gd 0.6 - 7. Field Oxide
e Active 0.6 - 45 8.
gd 0.04 - 9. Gate Oxide
gd Poly 0.4 - 10. Polysilicon
e NotPoly 0.44 - 45 11.
gd 0.07 - 45 12. 2nd Gate Oxide
gd Poly2 0.4 - 13. 2nd Polysilicon
e NotPoly 0.47 - 60 14.
gd 0.9 - 15.
e "P/P2/Act Contact"0.9- 60 16.
gd Metal1 0.6 - 17. Metal 1
e "Not Metal1"0.6- 45 18.
gd 1 - 19.
e Via 1 - 60 20.
gd Metal2 1.15 - 21. Metal 2
e "Not Metal2"1.15- 45 22.
gd 2 - 23. Overglass
e Overglass2 - 24.
7 Cell Hierarchy and Views

The cell is the basic building block of an integrated circuit design. A cell in L-Edit can contain three types of components:

- Primitives—geometrical objects created in the cell
- T-Cell code and parameters—instructions to generate layout in another cell (discussed in the chapter Parameterized Cells, page 374)
- Instances—references to other cells

In an efficient design, cells, primitives, and instances form a treelike hierarchical structure.

**Working with Cells in a Multi-User Environment**

Because OpenAccess is a multi-user database, before you can perform any operation that modifies a cell, you must reserve it. This prevents other users from editing or saving that cell view while you have it open. When you have a cell view reserved, no one else using the same library or database can modify that cell view.

In L-Edit, reservations are performed at cell level — you do not reserve an entire library as you would in S-Edit.

**Cell Views**

An OpenAccess cell is designated by its name, the library where it is saved and a view type.

Different view types can represent things like levels of abstraction in a design (behavior or gate), or different stages in the design process (post-synthesis, placed, or routed).

In L-Edit dialogs, cell names nomenclature is “cell name” “library” as shown below.
Cell Information

Use **Cell > Info** to edit information on the active cell. (See “Creating T-Cells” on page 374 for how to use the **T-Cells Parameters** tab.

You can view Cell Information without reserving a cell, but you must have the cell reserved in order to edit the cell information.

**Cell name**
Name of the active cell

**View name**
The cell view.

**Library**
The library to which the cell is saved.

**Cell data**
Includes **Author**, **Organization**, and **Information** (notes or messages) for the active cell. **Information** can contain a maximum of 256 characters.

**Cell version and dates**
The date and time the cell was created and last modified. The version numbering system provides an internal accounting method for tracking changes. **Increment** increases the version number to the next major version (for example, 3.08 to 4.00). L-Edit automatically increments the minor version number (for example, 3.08 to 3.09) each time changes to the cell are saved.

Use **Alt+K** to decrement version numbers (for example, 3.27 to 3.00 then 2.00).

Use **Alt+L** to increment the minor version (for example, 3.27 to 3.29).
Chapter 7: Cell Hierarchy and Views

The Libraries Navigator

L-Edit has a library and cell navigator similar to the one in S-Edit. The upper section lists the libraries that are open, with the top-level library in black text, and the lower section lists the cells in the selected library(s).

Columns in the cell list are sortable and configurable in several ways. You can click on a column header to sort the cell list, and you can click and drag on the column headers to size or remove them.
The Cell List Filter Menu

The pull-down menu for the cell list filter works two way. The **Filter** option lets you enter letters or numbers in the adjacent text-control field so that the cell list will only show objects that contain those characters, as they are entered.

The remaining options filter the cell list by the cell’s status in the design hierarchy. For example, “Decendants” will list only the cells and instances contained in the active cell.

If you apply more than one criterion using filters, a cell will be listed if it meets any of those criteria.

Note that filter settings are applied to all three setups – see “Saving Column Heading Setups 1, 2 and 3” on page 221.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Filter</strong></td>
<td>When you select this option, letters or numbers entered in the adjacent text-control field will filter the cell list to only show objects that contain the characters you have entered.</td>
</tr>
<tr>
<td>Top-level</td>
<td>Filters the list to show only those cells that are not instanced in any other cell.</td>
</tr>
<tr>
<td>Leaves</td>
<td>Filters the list to show only those cells that do not contain any instances themselves.</td>
</tr>
<tr>
<td>Children</td>
<td>Filters the list to show just cells that are instanced in the active cell (ie. one level down in the hierarchy).</td>
</tr>
<tr>
<td>Parents</td>
<td>Filters the list to show just cells in which the active cell is instanced (ie. one level up in the hierarchy).</td>
</tr>
<tr>
<td>Descendants</td>
<td>Filters the list to show cell instances (children) and leaves that are instanced in the active cell; i.e. every instance used in the cell.</td>
</tr>
<tr>
<td>Ancestors</td>
<td>Filters the list to show any cells in which the active cell is used. This includes all cells - parents, grandparents, great-grandparents, etc. - with the active cell below them in the hierarchy.</td>
</tr>
<tr>
<td>Modified</td>
<td>Filters the list to show just cells that have been modified since the database was last saved.</td>
</tr>
</tbody>
</table>

**Cell List Filter Settings**

**Settings > Filters** from the context menu lets you set global controls for how the cell list is filtered.
**Show Hidden Cells** toggles whether hidden cells are shown in the cell list, where a check sets them to be visible.

**Sub-string Filter** lets you set whether the filter field will be **Case-sensitive** and also which column the text-control field will apply to: **Filter All columns**, **Filter Cell-name** or **Filter View-name**.

**View-name Filter** opens a checklist showing all view names in the library, where a check means the view names will be shown in the cell list and no check means it will not. **Show All view-names** does just that. (You can also right-click on the **View** column header in the cell list to open the same checklist.)
How to Set Column Headings in the Cell List

To pick which columns will be displayed for each of the column setups, click on a setup number to make it active, then use the right-click menu in the cell list to select **Settings > Columns > Column Chooser**.

The column chooser shows all the headings that are not in use for a given column setup. Just click and drag headings from the chooser to the cell list header area.

<table>
<thead>
<tr>
<th>Cell</th>
<th>The cell name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>The cell view type.</td>
</tr>
<tr>
<td>Library</td>
<td>The library to which the cell is saved.</td>
</tr>
</tbody>
</table>
| Locked     | **Locked** indicates the cell is locked.  
(Blank) indicates the cell is not locked. |
| Status     | **Modified** indicates the cell was locally modified and is different from the version of disk.  
**Reserved** indicates the cell is reserved for editing, but is still unmodified (same as disk copy).  
(Blank) indicates the cell is not reserved and is unmodified. |
| **T-Cell** | T-Cell indicates the cell has dynamic Tanner parameters. (Blank) indicates a normal cell. |
| **Cache** | L-Edit uses a delayed read to save processing time, whereby it first reads a list of cells into the database and populates the cell navigator with their names and then when you open a cell, reads the cell contents and that of its instanced cells. The cache column indicates the various read states of a cell. |
| | • Cycle (applies only to a Memory cell) is artificially inserted to break cycle in a hierarchy. |
| | • Delay indicates the cell is loaded to the list but its contents are not yet in RAM. |
| | • Deleted means the cell is scheduled for deletion on the next save, but is not yet deleted on disk. |
| | • Memory indicates the cell exists in memory only. |
| | • Normal means the cell contents are loaded. |
| | • New means the cell is new and is not yet saved to disk. |
| | • ReadInProgress indicates the cell is being loaded from disk to RAM. |
| | • Unresolved indicates the cell is referenced but is not found on disk. |
| **LastOp** | Indicates the outcome of the most recent operation attempted on the cell. Possible values are Success, Skipped or Error. |
| **LastOp Comment** | Describes the most recent operation completed on the cell. For example, “Cell 'name' saved.” or “Update is not applicable, because cell is up-to-date.” |

**Saving Column Heading Setups 1, 2 and 3**

Column headings in the cell list are user-configurable and “sticky” - they are saved and will be the same until you change them. Setup tabs 1, 2 and 3 allow you to identify and alternate between three pre-set column configurations. Click on one of the numbers to use a given setup. You can also select a setup using (right-click in cell list) > Settings > Columns > Setup 1, 2 or 3.
Setting Colors in the Cell List

You can right-click in the cell list and choose **Settings > Color** to toggle whether certain cell states will be indicated with a colored background or text. A check turns color highlighting on, no check turns it off.

**Filtered List**

- **When a list is filtered**, the entire list has a light yellow background.

**Reserved Cells**

- The row of a reserved cell will have a light green background.

**Modified Cells**

- The row of a cell that has been modified but not saved will have a light red background.

**Hidden Cells**

- When **show hidden cells in lists** is set to “on,” the name of a hidden cell is shown in gray.

**Unresolved Cells**

- When a cell is referenced but not found on disk, the name of the cell is shown in red.

Highlighted vs Selected in the Cell List

Although you can *select* multiple cells in the cell list, only one of those selected cells is the one *highlighted*. This distinction is important because many operations can be launched with multiple cells selected, but the operation will only apply to the cell that is highlighted. L-Edit provides a slight visual indicator by enclosing the highlighted cell in a dotted line, as shown below for cell `inv`.
If multiple instances are selected, the referenced cell of the first instance in the selection will be highlighted. If no instance is selected, the last cell opened will be highlighted.

**Right-Click Menu in the Cell List**

The right-click menu in the cell list provides a shot cut to most cell and instance operations.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>See “Creating a New Cell” on page 227.</td>
</tr>
<tr>
<td>Open</td>
<td>Opens an existing cell view. For a generated (T-Cell) cell, you will have</td>
</tr>
<tr>
<td></td>
<td>the option to Open Layout or Open Code. See “Opening a Cell” on page 229.</td>
</tr>
<tr>
<td>Instance</td>
<td>See “Instancing Cells” on page 249.</td>
</tr>
<tr>
<td>Update</td>
<td>Retrieves the cell from its library of origin and updates the local version.</td>
</tr>
<tr>
<td></td>
<td>If you have unsaved modifications to the cell, L-Edit will prompt you to</td>
</tr>
<tr>
<td></td>
<td>revert changes or save them instead. See “Updating a Cell” on page 229.</td>
</tr>
<tr>
<td>Save</td>
<td>See “Saving a Cell” on page 231</td>
</tr>
<tr>
<td>Copy</td>
<td>See “Copying a Cell” on page 236</td>
</tr>
<tr>
<td>Rename</td>
<td>See “Renaming or Moving a Cell” on page 231.</td>
</tr>
<tr>
<td>Delete</td>
<td>See “Deleting a Cell” on page 239.</td>
</tr>
<tr>
<td>Replace Instances</td>
<td>See “Replace Instances of This Cell” on page 263 and “Replace Instances in Selected Cells” on page 263.</td>
</tr>
</tbody>
</table>
Chapter 7: Cell Hierarchy and Views

Reserving a Cell

Before you can perform any operation that modifies a cell view, you must reserve it. This prevents other users from editing or saving that cell view while you have it open. When you have a cell view reserved, no one else using the same library or database can modify that cell view.
While a cell view is reserved, L-Edit places a file with extension `.cdslck` in the folder for the cell view, and the background of the cell row in the Libraries navigator will be shaded light green.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Data Mod</th>
</tr>
</thead>
<tbody>
<tr>
<td>data.dm</td>
<td>DM File</td>
<td>6/5/2012 2:</td>
</tr>
<tr>
<td>layout.dae</td>
<td>OA File</td>
<td>6/5/2012 2:</td>
</tr>
<tr>
<td>layout.dae.cdslck</td>
<td>CDSLCK File</td>
<td>6/5/2012 2:</td>
</tr>
<tr>
<td>master.tag</td>
<td>TAG File</td>
<td></td>
</tr>
</tbody>
</table>

**Explicit Reservation**

You can reserve a cell explicitly either by pressing the **Toggle Reserve** button on the Database toolbar (depressed indicates that a cell is reserved), or by selecting a cell or multiple cells in the Libraries navigator and using the right-click menu to select **Status > Reserve**.

When you create a new cell view it is automatically reserved.

However, opening an existing cell view does not automatically reserve it. Therefore, you may want to explicitly reserve it in anticipation of making edits. Similarly, saving a cell or closing a cell does not automatically unreserve it.

If you attempt to explicitly remove a reservation from a cell view with unsaved changes, L-Edit will prompt you to **Save** all edits made since the last time the cell was saved, **Revert** the cell to its status when it was reserved, or to **Cancel** without performing any action.
You can also use Select Hierarchy from the right-click menu in the Libraries navigator to select all cells instanced in the highlighted cell so they can be simultaneously reserved.

**The Database Toolbar**

The Database toolbar buttons provide shortcuts for saving cell changes to the and updating cells from the database.

- **Toggle reserve of cell view**
  - Press this button to reserve a cell explicitly, where the depressed state indicates that a cell is reserved.

- **Save cell**
  - Press this button to save the active cell.

- **Save all**
  - Press this button to save all open cells.

- **Update cell**
  - Press this button to check the database for changes to cells stored on disk. Any cell in your design that has been changed on disk and that you do not have reserved will be updated with the newer version. Cells you have reserved are not updated.

- **Update all**
  - Press this button to update all open cells.

**Implicit Reservation**

If you attempt to modify a cell without explicitly reserving it, L-Edit will attempt to reserve the cell on your behalf. If it is not reserved, as soon as you make a modification L-Edit will place it in the reserved state so you can continue to edit and save.

If the cell view is reserved by another user, L-Edit will give you the option to view related information.
If you select Yes to see that information, L-Edit will display the Reservation Details dialog shown below.

---

**Reservation Details**

```plaintext
# Edit Lock-Store file. CAUTION: Please do not change.
#
# Information about current Edit Lock Owner.
#
LockStepVersion: 1.1
LoginName: qa009xp.tanner.com
HostName: 4040
ProcessCreationTime.UTC: 1339513817
AppIdentifier: QA File System Design Manager
OSType: win32
ReasonForPlacingEditLock: OpenAccess edit lock
FilePathUsedToEditLock: \raid\srv02\ED\Engineering\QA\JudyTest\v16DocExample\RingWCC
TimeEditLocked: 6/12/2012 8:10:45 AM Pacific Daylight Time
```

---

### Unreserving a Cell

After you have made modifications to a cell and saved your changes, you should Unreserve the cell so other users may edit that cell. To unreserve a cell, press the **Toggle Reserve** button on the Database toolbar, or uncheck **File > Database > Reserve Cell**.

When a cell is Unreserved, the .cdslck file from the folder for the cell view is removed. If an Unreserve is attempted on a cell with unsaved changes, you will be asked if you want to Save, Revert or Cancel:

- **Revert** will discard your changes, re-read the cell from the database, and release the reservation. **Cancel** closes the dialog and returns to the editing session.

### Listing of Reserved Cells

The command **File > Database > Cell Reservation Report**, can be used to create a report showing which cells are reserved, each cell's reservation owner and the time the reservation was made.

### Creating a New Cell

To create a new cell, choose **Cell > New** or press N. The **General** tab allows you to enter identification information about the cell. If the cell will contain only primitives and instances, click **OK**. To add
parameters and UPI code for a T-Cell, click the **T-Cell Parameters** tab (see “Creating T-Cells” on page 374).

Name of the new cell. L-Edit assigns the default name “cellx” where x is an integer that increments automatically with each new cell.

View The cell view.

Library The library to which the cell is saved.

Cell info Enter the **Author**, **Organization**, and **Information** (notes or messages) for the new cell.

Open in new window A check instructs L-Edit to open the cell in a new layout window. The **Select Cell To Edit** dialog (accessed with **Cell > Open**) has an identical option. Setting this option in either dialog controls the behavior for both commands. L-Edit saves the last state of this check box when you exit the application.

When you create a new cell view, it is automatically reserved, and a folder for that cell is immediately created on disk so that there is a place to put the reservation file. The new cell will remain reserved until you explicitly unreserved it.
Opening a Cell

To open a cell, choose Cell > Open, press O, or click the open cell button ( ).

Note: Note that opening an existing cell view does not automatically reserve it.

Library
The name of the current file (default) or of any other open file.
The specified file’s component cells are displayed in the scrollable list. To open a cell, select it and click OK or double-click it.

Show hidden cells
When checked, shows all cells in the list, including hidden cells.
When unchecked, does not show hidden cells. You can make a cell hidden by deselecting the Show in Lists option in Cell > Info—Cell Information. Cells generated by a T-Cell are automatically hidden by L-Edit.

Filter

Open View
Instructs L-Edit to open either the cell Layout or a text window containing T-Cell Code for the selected cell. If the selected cell is not already a generator cell, the option T-Cell Code will open a blank code window.

Updating a Cell

To update a cell, L-Edit checks the database for changes to the cell stored on disk. Any cell in your design that has been changed on disk and that you do not have reserved will be updated with the newer
Cells you have reserved are, of course, not updated. Note that a user must decide when and whether to check for cell updates.

If you attempt to update a cell that has unsaved changes, L-Edit will give you the option to Revert edits, which tells L-Edit to proceed with the update. The unsaved changes will be lost. Cancel closes the dialog and returns to the editing session without changing the cell.

You can update a cell at any time by pressing the Update cell button on the Database toolbar ( ), or by selecting a cell or multiple cells in the Libraries navigator and invoking Update from the right-click menu.

**Note:** A cell update must be self-initiated — L-Edit does not provide an alert when a library cell is changed.

When a cell is updated, L-Edit logs a message to the command line window describing the changes that were made, or noting that the cell was already up to date.

You can update all cells in a design as well as any technology changes from the database using the Update all button on the Database toolbar ( ).

When performing Update All L-Edit will silently skip cells with unsaved changes and will not revert them to the version on disk. Also, the list of cells will be updated with any new cells created by other users, and cells that have been deleted by other users will be removed from your list of cells.

If you are editing in place when you do an Update all, you will be popped out to the top cell.

### Reverting a Cell

**Cell > Revert Cell** allows you to reverse all layout changes to the active cell after any the following operations. However, the Revert Cell command does not reverse changes to T-Cell code.

- **File > Save**
- **Tools > Generate Layers**
- **Tools > DRC**
Saving a Cell

You must have a cell reserved to save any changes to that cell.

To save changes to the active cell, press the **Save Cell** button on the Database toolbar. To save changes to multiple cells, select the cells in the Libraries navigator and invoke Save from its right-click menu.

To save your changes to all cells, press the **Save all** button on the Database toolbar, which is identical to the Save all button on the Standard toolbar. When **Save all** is invoked, the cells that were saved are logged to the command line window.

**Note:** Saving one or all cells does not release the reservation on a cell — you must do so explicitly.

When an attempt is made to modify a cell or to reserve a cell, L-Edit will check that the cell is not reserved. If it is not reserved, L-Edit will next check if any modifications to that cell have been made by another user. If modifications to the cell have been made by another user, L-Edit will indicate as shown in the dialog below that modifications have been made, and will re-read that cell from the database to obtain the latest updates.

Renaming or Moving a Cell

The **Cell Rename/Move** operation copies a selected cell to either a new cell name, a different library, or both a new cell name and library location.

Once a cell is relocated, L-Edit deletes the old name, removes the cell from its previous library, or both.

Invoke **Cell > Rename**, right-click on one cell in the Libraries navigator and select **Rename**, or use the shortcut keys **Shift + R**.

The **Cell Rename/Move** operation will also retarget instances of the original cell to the newly named/moved cell. For OpenAccess databases, this requires loading of cells, which can be a time consuming operation. Limiting the list of libraries in which to check for instances, for example by excluding PDK libraries in which such instances are known not to exist, can improve performance of this operation.
When you press OK in the Cell Rename/Move dialog, prior to renaming or moving the cell, L-Edit will analyze the database to determine if the required reservations can be made, whether a cell with the target cell name already exists, and if the libraries or cells involved are locked. (See “Deleting a Cell” on page 239.)

After the move or rename operation the new cell is open and active.

![Image of Cell Rename/Move dialog]

**To cell name**
The new name of the active cell.

**To library**
The new library location of the active cell.

**Retarget instances of cell found in:**
Select the list of libraries in which to check for instances of the renamed/moved cell. This field is only present for OA databases. For tdb databases, all libraries are checked.

Before netlist extract, it is possible to have two instances with the same instance name. However, after netlist extraction, L-Edit will force each instance to have a unique name.

**Closing a Cell with a New Name**

You can also use **Cell > Close As**, which saves the active cell with all current changes to a new view, and simultaneously closes the original without saving any of the changes since the previous save operation.
After the close cell as operation the new cell view is open and active.

**Saving a Single Cell to Another File**

Use **File > Save Cell to TDB File** in either an OpenAccess or TDB database to save a single cell and all its hierarchy intact to a new TDB file. This is helpful for sharing or distributing work.

**Copying a Piece of a Cell to Another Cell**

The command **Draw > Clip Out Region**... copies a rectangular region of a cell and saves that region and its hierarchy to a new cell. All geometry within the boundary of the clipped rectangle and all instances completely inside the clip rectangle will retain their hierarchy. Objects on hidden layers will be copied.
The **Clip Out Region** dialog shown below lets you name the new cell and specify the portion to be copied.

![Clip Out Region](image)

**New Cell**
- Enter the Name and View type for the new clipped region. L-Edit appends “_clipout” to the name of the source cell as the default if nothing else is entered in this field.

**Clip To**
- Define the rectangle to be copied, either by entering absolute x, y coordinates for the corners, or by clicking the **Pick...** button.

**Pick...**
- Click on this button to use the mouse to select the rectangular area that will be clipped. The first click selects the lower left point, the second click selects the upper right point of the rectangle to be copied.

**Database Evaluation When Copying or Deleting Cells**

Prior to performing most cell operations, L-Edit must analyze the database to determine if the cells involved are available for the requested operation.

For example, when you attempt to copy a cell and its hierarchy, some cells may need a reservation, some cell views with the target name may already exist, some of the libraries or cells may be locked, etc. Because any or all of these obstacles may be the case, L-Edit must perform a comprehensive sequence of queries and permission acquisition prior to performing an operation.

L-Edit checks the database using the following chain of considerations.

<table>
<thead>
<tr>
<th><strong>Requested Operation</strong></th>
<th><strong>Conditions</strong></th>
<th><strong>Result</strong></th>
</tr>
</thead>
</table>
| Create a target cell view in a target library using **Cell > Copy** or **Cell > Rename/Move** | Target cell can be created in the target library if:  
  - Target cell does not exist in target library.  
  - Target library is not locked. | Target cell is created in target library. |
(Because copying a cell is the first step in the rename/remove operation, L-Edit performs the same evaluations for both operations.)

<table>
<thead>
<tr>
<th>Copy cell hierarchy (Cell &gt; Copy &gt; Copy Hierarchy)</th>
<th>Target cell cannot be created in the target library because:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Target cell exists but is reserved.</td>
</tr>
<tr>
<td></td>
<td>- Target cell exists but is locked.</td>
</tr>
<tr>
<td></td>
<td>- Target library is locked.</td>
</tr>
<tr>
<td></td>
<td>L-Edit opens a dialog stating that target cell could not be created, and copy operation is aborted.</td>
</tr>
<tr>
<td></td>
<td>L-Edit opens a dialog stating that target cell could not be created, and copy operation is aborted.</td>
</tr>
<tr>
<td></td>
<td>Target cell exists in target library, but can be overwritten. This requires all of the following:</td>
</tr>
<tr>
<td></td>
<td>- Target cell is not reserved by another user.</td>
</tr>
<tr>
<td></td>
<td>- Target cell is not locked.</td>
</tr>
<tr>
<td></td>
<td>- Target library is not locked.</td>
</tr>
<tr>
<td></td>
<td>L-Edit opens a dialog asking to allow overwrite.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Copy hierarchy is checked, and target library is different than source library; and targets of hierarchy cells exist in the target library but can be overwritten. This requires all of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Target cell is not reserved by another user.</td>
</tr>
<tr>
<td>- Target cell is not locked.</td>
</tr>
<tr>
<td>- Target library is not locked.</td>
</tr>
<tr>
<td>L-Edit opens dialog asking to allow overwrite or not.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Retarget instances are checked, and all cells containing instances of the source cell can be modified. This requires all of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Cells containing instances of source cell are not reserved by another user.</td>
</tr>
<tr>
<td>- Cells containing instances of source cell are not locked.</td>
</tr>
<tr>
<td>- Library containing cells with instances of source cell is not locked.</td>
</tr>
<tr>
<td>L-Edit renames cell instances from &lt;source cell name&gt; to &lt;target cell name&gt; in all libraries.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Retarget instances are checked, but some cells containing instances of the source cell cannot be modified, due to any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Cells containing instances of source cell is reserved by another user.</td>
</tr>
<tr>
<td>- Cells containing instances of source cell is locked.</td>
</tr>
<tr>
<td>- Library containing cells with instances of source cell is locked.</td>
</tr>
<tr>
<td>L-Edit opens a dialog asking if copy should proceed.</td>
</tr>
</tbody>
</table>

| If you select continue, instances of the source cell in cells that could not be modified will continue to point to source cell. |
| All others will be modified to point to the new cell. |
Copy a Cell

You can copy cells within a design or to another design. In the copy cell process, L-Edit creates a new cell, not an instance, including all primitives and instances defined by the original cell. When a cell is copied to another file, all cell definitions of the instances in the copied cell are also copied.

The Copy Cell command allows copying a cell from one library to another within the same database, or from one database to another, with options to copy hierarchy, retarget instances of the old cell to the new cell, and delete originals.

(L-Edit can also copy just a portion of a cell and paste it into a new cell created for that “clip out;” see “Copying a Piece of a Cell to Another Cell” on page 233.)
Choose **Cell > Copy**, press **C**, click the copy cell button ( ), or right-click on the cell in the Libraries navigator and select Copy.

**Cell Copy**

- **To cell name**: Enter the new cell name.
- **To database**: Click the **Change** button to select a different database as the target of the copy.
- **To library**: For an OpenAccess design, select the active library (default) or any other open library from the drop-down list. The Library field will be disabled for TDB databases, as only the toplevel library can be modified in a TDB database.
- **Copy hierarchy**: If checked, copies the full hierarchy of source cells to the target library, if target library is different to source library, and updates instances in the target cell to reference the copied hierarchy.

**Retarget instances from**

- **source cellname:library** to **target cellname:library**

If checked, updates all instances of the source cell to point to the target cell, in all libraries.

If not checked, instances of the source cell will continue to point to the original source cell.

Note that this operation reads all cells in all libraries, so it may run slowly in large OpenAccess databases.
Copying to a different database

To copy a cell to a different database, press the **Change** button in the **Cell Copy** dialog.

**Change Target Database**

**Target database:**

- ADC8

**Match layers by:**

- **Name**
- **GDS number**
- **OpenAccess layer number**

**For unmatched layers:**

- **Create layer in target database**
- **Ignore geometry on unmatched layer**

**Match layers by**

- **Name** — Objects in source database are copied to target database on same layer name.
- **GDS Number** — Objects in source database are copied to target database on layer with same GDS Number.
- **OpenAccess layer Number** — Objects in source database are copied to target database on same OpenAccess layer number.

**For unmatched layers**

- **Create layer in target database** — If matching layer in target database does not exist, then create a new layer.
- **Ignore geometry on unmatched layer** — If matching layer in target database does not exist, then ignore geometry on source layer.
Deleting a Cell

To delete a cell or cells choose **Cell > Delete** or right-click in the Libraries navigator and select Delete. If you do not select any of the checkboxes, L-Edit will simply delete the selected cell(s). If you use one or more of the checkbox options, L-Edit performs those actions in addition.

The **Cell Delete** operation includes options to check for instances of the cell to be deleted. For OpenAccess databases, this requires loading of cells, which can be a time consuming operation. Limiting the list of libraries in which to check for instances, for example by excluding PDK libraries in which such instances are known not to exist, can improve performance of this operation.

Delete hierarchy

This option delete all cells instanced in the selected cell as well as the cell itself. L-Edit will first attempt to get a reservation on these cells, and will warn as follows if it cannot obtain all necessary reservations:

Selecting **Proceed** will delete instances for which the containing cell could be reserved. Instances in which the containing cell could not be reserved are left as unresolved instances. Selecting **Cancel** cancels the Delete operation and releases any reservations that had been obtained.

Delete cell even though it is locked

This option operates independently of the other two options - the delete behavior with respect to reservations remains the same, but even locked cells will be deleted. Note that when a cell is deleted, it is marked as hide in lists but it is not removed from disk until a Save all is performed.
Chapter 7: Cell Hierarchy and Views

Deleting a Cell

Delete instances of this cell
Deletes instances of the selected cells in selected libraries. L-Edit will attempt to get a reservation on the cell in every library, and will warn as follows if a reservation cannot be obtained:

Preserve cell if it is instanced
When this option is enabled, you will not be able to delete a cell if it is instanced in any other cell. If the Delete hierarchy option is also enabled, only those cells that are not instanced somewhere else will be deleted.

Check for instances of cells in
Specify the list of libraries in which to check for instances of the cell to be deleted, for the Delete instances of this cell and Preserve cell if it is instanced options. This option is present for OpenAccess databases only. For TDB databases, all libraries are checked.
## Listing the Object Types or Layers Used in a Cell

**Cell > Cell Object Summary** counts the type and number of objects on each layer used in a cell, for the top level of the cell and all objects in the hierarchy. Each object is counted only once.

### Current Cell - Cell1 - Layer Object Statistics

<table>
<thead>
<tr>
<th>Layer</th>
<th>Box</th>
<th>Polygon</th>
<th>Wire</th>
<th>Circle</th>
<th>Pie</th>
<th>Tors</th>
<th>Port</th>
<th>La</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Instances**: 4

### Current Cell & Hierarchy - Cell1 - Layer Object Statistics

Objects in cells down the hierarchy are only counted once.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Box</th>
<th>Polygon</th>
<th>Wire</th>
<th>Circle</th>
<th>Pie</th>
<th>Tors</th>
<th>Port</th>
<th>La</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Contact</td>
<td>43</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Metal1</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>Metal2</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>N_Implant</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P_Implant</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Poly</td>
<td>35</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>SilicideBlock</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Via1</td>
<td>58</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>275</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

**Instances**: 4
Cell > Layer-Cell Cross Reference

**Cell > Layer-Cell Cross Reference** lists the layers in a design file that have geometry on them and gives the name of the cells that contain the geometry.

```
Layer-Cell Cross-Reference
******************************************************************************
Layer "Active"
  Cnt_active
  Cnt_HTAP
  CurrentMirror_NMOS2S_Auto_1.5u_0_1_0_250n_1_1_1_0_0_0_0_0.650
  NMOS2S_Auto_0_1.5u_0_250n_1_1_0
******************************************************************************
Layer "Contact"
  Cnt_active
  Cnt_HTAP
  Cnt_Poly
  Cnt_PTAP
  CurrentMirror_NMOS2S_Auto_1.5u_0_1_0_250n_1_1_1_0_0_0_0_0.650
  NMOS2S_Auto_0_1.5u_0_250n_1_1_0_0
  rnp_auto_0_0_1_25u_0.650_0_1_u_0_0.650_0_0
******************************************************************************
Layer "Metal1"
  Cnt_active
```

The Design Navigator

The Design Navigator lists all cells in a design in a hierarchical structure, including information on instances and the fabrication cell. To open the Design Navigator, use **View > Design Navigator**, or click **** in the standard toolbar. You can also use **View > Design Navigator** (shortcut **F10**) to simultaneously open the Design Navigator and arrange all open windows, with the Design Navigator docked on the left and the layout windows, each the same size, stacked.

You can double-click on a cell name to open a layout view of that cell. Note that L-Edit will open a separate layout window for a given cell each time you double-click on the cell name in the list.
The Design Navigator always opens in **Top down - all cells** display mode. The default list order is alphabetical. You can also sort by the date cells were modified or their DRC status.

You can use the search field to find a cell more quickly. When you type letters in this field, L-Edit finds and highlights the first name in the list that begins with the characters entered.

Use this drop-down list to select from the following display modes:
- “Top down - all cells” on page 245
- “Bottom up - all cells” on page 245
- “Top down - non-instanced” on page 246
- “By date modified” on page 246
- “DRC Status” on page 247

Collapses all cells to display only the top level.

Expands all cells to display all levels of instancing.

Creates a new cell.

Deletes the selected cell. This operation is only available when the selected cell is not instanced in another cell.
Use the **Copy To Text View** button to create a text version of the Design navigator display, which will launch as a log file in the L-Edit text window. You can also use the menu command **File > Print Preview** to toggle the Design navigator view to a text version, as shown below.

The **Close** button returns to the conventional Design navigator display.

Shows all cells in the list, including hidden cells.

**Design Navigator Display Modes**

**Note:** Note that autogenerated cell instances are displayed in the Design Navigator below their generator cell.
**Top down - all cells**

In this mode cells are hierarchically listed in terms of instances they contain. When you expand the cell, the instanced cells are listed below it. A number in brackets indicates the number of times a cell is instanced in the higher cell.

---

**Bottom up - all cells**

In this mode cells are listed in terms of where they are instanced. When a cell is instanced in other cells it is marked with a + and the cells which contain it as an instance are listed below it. A number in brackets indicates how many times the higher-level cell is instanced in the particular cell.
**Top down - non-instanced**

This mode display only the cells that are not instanced in other cells.

![Top down - non-instanced](image)

**By date modified**

This mode displays all cell sorted by the date and time they were last modified, from most to least recently. Bold text indicates a cell that has been modified but not saved. The version is displayed for reference only and subcells are not displayed in this mode.

![By date modified](image)
DRC Status

In this display mode, all cells are listed in alphabetical order with an icon indicating their DRC status (see “DRC Status” on page 554.)

DRC Status Icons

- **DRC Needed.** Either DRC has not been run on this cell, changes have been made to this cell since the last DRC run, or the DRC setup has changed since the last DRC run.
- **DRC Failed.** DRC has been run on this cell, and errors were found.
- **DRC Passed.** DRC has been run on this cell and no errors were found.

Excluding all instances of this cell from DRC.

Icons in the Design Navigator

L-Edit uses these symbols to indicate a cell’s status. If a cell name is in bold type, it indicates the cell has unsaved modifications.

- The padlock indicates a locked cell.
  - In the **Select Cell to Delete** dialog, a cell that cannot be deleted.
  - In the **Select Cell to Instance** dialog, a cell that cannot be instanced.
- A locked cell that cannot be deleted or instanced.
- The fabrication cell.
- A locked fabrication cell.
- A fabrication cell that cannot be deleted or instanced.
- A locked fabrication cell that cannot be deleted or instanced.
- The generator icon indicates a T-Cell.
- A locked T-Cell.
Performing Cell Operations with the Design Navigator

To access the context-sensitive menu of available commands for the Design Navigator, select a cell and click the right mouse button.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Opens the selected cell.</td>
</tr>
<tr>
<td>New</td>
<td>Creates a new cell.</td>
</tr>
<tr>
<td>Rename</td>
<td>Renames the selected cell.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the selected cell. This command is only available when the selected cell is not instanced in another cell.</td>
</tr>
<tr>
<td>Lock</td>
<td>Locks/unlocks the selected cell.</td>
</tr>
<tr>
<td>Hierarchical Lock</td>
<td>Locks the selected cell and all cells that are instanced in it (see “Locking and Unlocking Cells Hierarchically” on page 265).</td>
</tr>
<tr>
<td>Hierarchical Unlock</td>
<td>Unlocks the selected cell and invokes a warning dialog that asks if the user wants to individually unlock each cell that is instanced in it. The dialog also gives the option to unlock all instanced cells.</td>
</tr>
<tr>
<td>Flatten</td>
<td>Flattens the hierarchy of the selected cell.</td>
</tr>
<tr>
<td>Fabricate</td>
<td>Marks the selected cell for fabrication.</td>
</tr>
<tr>
<td>Save Cell to TDB File</td>
<td>Opens a dialog to save the selected cell to a TDB file. The default filename is CellName.tdb.</td>
</tr>
<tr>
<td>Create/Update T-Cell Code...</td>
<td>Creates a new T-Cell code template for the selected cell. If the cell already has T-Cell code, L-Edit displays a warning and asks if you want to replace the existing code.</td>
</tr>
<tr>
<td>Info</td>
<td>Displays the Cell Information dialog for the selected cell.</td>
</tr>
</tbody>
</table>
Copying and Instancing from the Design Navigator

If a cell belongs to a library used by both designs, you can copy or instance cells from one design to another using drag-and-drop operations from the Design Navigator:

- If you drag and drop a cell from the Design Navigator onto the layout of the same file, L-Edit creates an instance of the cell.
- If you drag and drop a cell from the Design Navigator to the Design Navigator of another file, L-Edit creates a copy of the cell in the other file.
- If you drag and drop a cell from the Design Navigator onto the layout of another design, L-Edit creates an instance of the cell in the other file. If you right-click and drag, you will have the option to either instance, copy and instance, or copy the cell, as shown below.

Instancing Cells

An instance is a representation of a cell in a particular location and orientation in another cell. An instance can reference a cell composed of primitives, other instances, or a combination of primitives and instances. An instance can also reference a cell generated from T-Cell code, called an auto-generated cell. However, a cell cannot instance itself—that is, you cannot create an instance of a cell in the cell itself.

An “instancing” cell contains such a representation; an “instance” cell is the source, or referenced, cell. Changes made to the instanced (source) cell are automatically propagated to all instances of that cell. Changes made to a T-Cell, which L-Edit uses to generate source cells, cause all auto-generated cells and their instances to be flagged “out of date.” You can update T-Cell instances using the Cell > Regenerate T-Cells command.

Layouts that use instances consume less memory than “flat” designs, where all the features exist as originally drawn objects.

Toggling Instance Selectability

Note that it is possible to set L-Edit so that instances can or cannot be selected. This setting applies globally, to all instances in a design. When instance selection is not allowed (command unchecked), the icon will have a red dot in the lower left corner ( ).
However, if you place a new instance while instances are unselectable, it be in the selected state and stay that way until deselected so it can be positioned. If you accidentally unselect it, **Undo** will return it to its previous position and restore the state of selection.

Use **Edit > Allow Instance Selection** from the menu bar or right-click on the **Instance** button (_drag) in the Drawing toolbar to access the same command.
Creating Instances

To instance a cell, use **Cell > Instance**, press **I**, or click on the **Instance** icon in the Drawing toolbar to open the **Select Cell to Instance** dialog. You can also drag and drop a cell from the Design Navigator into the current file.

**Select Cell to Instance**

- **Library**: Name of the library from which cell to instance can be selected.
- **Filter**: Use the **Filter** field to narrow the displayed list. When you type a letter in the **Filter** field, L-Edit displays only those cell names containing that character. When you type a second letter, L-Edit displays only those cell names containing those two characters in that same order, and so on.
- **Instance Name**: Name of the instance currently selected in the cell list. To select a cell, highlight its name in the cell list. Double-click or click **OK** to create an instance of the cell.

In the cell list, bold font indicates that a cell has been edited but the changes have not been saved.

**Note**: Each instance in a cell must have a unique name after netlist extraction. Before netlist extract, you can have two instances with the same instance name. However, after netlist extraction, L-Edit will force each instance to have a unique name.
Replace

Replaces the instance in the layout area with an instance of the specified cell. See “Replacing Multiple Instances using Edit Object(s)” on page 265.

When this option is checked, two other options become available:

- **Abutment**—aligns the instance selected in the layout with the replacement instance according to their abut ports. For a detailed description, see “Aligning Instances by Abut Ports” on page 262.

- **Origin**—aligns the instance selected in the layout with the replacement instance according to their origins. With this option, the replaced instance maintains the position of the previous instance with respect to the origin (position 0,0) of the coordinate system.

Show hidden cells

When checked, shows all cells in the list, including hidden cells. When unchecked, does not show hidden cells. You can make a cell hidden by deselecting the **Show in Lists** option in **Cell > Info—Cell Information**. Cells generated by a T-Cell are automatically hidden by L-Edit.

If you have selected an instance in the layout of a cell that is not set to **Show in lists** in **Cell > Info**, you must enable **Show all cells** to be able to instance that cell.

**Editing Instances**

With certain exceptions, to edit the contents of an instance you must edit the cell it refers to.

From the layout, a cell instance cannot be reshaped, sliced, or merged; and vertices and edges cannot be individually edited.

The following operations can be performed on an instance in the layout window:

- **Move** (see “Repositioning” on page 321)
- **Rotate or flip** (see “Reorienting” on page 326)
- **Edit using data values** (see “Reorienting” on page 326)
- **Replace** (see “Replacing Instances” on page 261)

An instance cannot be edited or moved if it contains objects that are drawn on a locked layer. To edit or move such an instance, you must first unlock any locked layers.

You can edit the contents of an original layout cell, or auto-generated T-Cell instance or array in two ways:

- **Open the original cell and make the desired changes there**
- **Use **Edit > Edit In-Place** (see “Editing an Original Instance In-Place,” below.)**

Changes made to an original layout cell or auto-generated T-Cell are automatically propagated to all instances and arrays of that cell.
Editing an Original Instance In-Place

*Editing in-place* allows you to edit the source cell of an instance by editing one of its instances.

To edit an instance in place, select the instance and use **Edit > Edit In-Place > Push Into**, or the shortcut **Page Down**, or the edit in-place button ( ) to “step down” into the instance.

Once you have opened an instance in place, the edits you make are equivalent to editing the instance’s source cell. Note that editing in-place is not available for instances that have been rotated by a non-orthogonal angle.

Editing a T-Cell instance in-place allows you to edit the contents of the auto-generated cell corresponding to the selected instance, and automatically propagates the changes to all instances of the T-Cell that were created with the same parameters.

While editing in-place you can only select or edit objects contained in the instance. This includes regular geometry as well as other instances or arrays. Editing in-place does not allow you to change T-Cell code.

To step down multiple levels in an instance, continue selecting instances and using **Edit > Edit In-Place > Push Into** as described above.

To step up in the hierarchy and end the edit in-place session, use **Edit > Edit In-Place > Pop Up**, the shortcut **Page Up**, or click the pop up edit in-place button ( ) on the Standard toolbar.

When you are editing in-place, you can use **Edit > Edit In-Place > View Top Cell** or press the **End** key to move selection to the home view of the top cell.

Use **View > Home** or press the **Home** key to go to the home view of the cell currently being edited in the instance hierarchy.

**“Push to Object”**

You can step through the design hierarchy multiple instances at a time using **Edit > Edit In-Place > Push to Object** (shortcut **Ctrl+PageDown**). **Push to object** moves the editing context though multiple hierarchy levels to select the object that the mouse pointer is currently over. Invoking **Push to Object** repeatedly will cycle selection through the objects the cursor is over, moving up and down the hierarchy as necessary.

Use **Edit > Edit In-Place > Pop to Top Cell** (shortcut **Ctrl+Page Up**) to move all the way up the hierarchy to the top cell.

---

**Note:**

Changes made to a T-Cell instance using **Edit > Edit In-Place** will be lost if you regenerate the T-Cell.
Edit Object(s) for Instances

To edit an instance as an object, choose the **Edit Object(s)—Instances** tab. You can change the name of the instance and factors that affect the display of the instance.

**Instance of cell**

Name of the instanced cell. Disabled when multiple instances are selected.

To instance a different cell, click the **Change** button to open the **Select cell to instance** dialog.

**Instance name**

Identifies the selected instance or array. Disabled when multiple instances are selected. L-Edit automatically assigns a name if you leave this field blank.

*Note:* Each instance in a cell must have a unique name after netlist extraction. Before netlist extract, you can have two instances with the same instance name. However, after netlist extraction, L-Edit will force each instance to have a unique name.
Coordinate system

Sets two options controlling the rotation of selected instances and their coordinate system.

- **Rotation angle**—the angle by which the instance is rotated. The coordinate axes illustration is updated as the angle is changed. Coordinates of arrays are specified with respect to the instanced cell. If the database resolution is insufficient for accurate rendering of the rotated instance, L-Edit provides a warning that recommends the subgrid be rescaled.

- **Mirror**—when checked, flips the instance coordinate system horizontally. The coordinate axes illustration reflects the change.

Translation (*Display units*)

The position of the instance with respect to the origin of the instancing cell. When you first create an instance, L-Edit places it at the center of the visible layout area. Moving the instance changes the x- and y-coordinates.

Scale factor

A fraction that defines the scaling of the instance relative to the original cell. This factor is applied to the X and Y coordinates of all objects in the instanced cell. Scaled instances maintain their proportions and geometry in both GDSII and CIF formats, but CIF output results in the creation of new cells which are scaled versions of the originals.

Array parameters

Enter number of times the instance is arrayed in the X and Y directions of the instanced cell’s coordinate system.

Array parameters

Enter the X and Y spacing, respectively, between array elements.

Array parameters

Press either or both of these buttons to set the corresponding ΔX or ΔY field to the MBB of the selected instance.

Locked

When this option is checked, an instance is locked so that it cannot be modified by any editing operation.

This is useful, for example, when you want to edit other parts of a cell but keep one specific instance in a fixed position or impossible to change in any other way.

An instance’s locked state is persistent in a TDB file, and is saved as a property of an instance or via an OA database.

(Note that the a checkbox in the Setup Application–Selection dialog globally toggles both selection of locked instances and selection of objects on locked layer.)

**Note:** Instances are not affected by **On layer** or **GDSII Data type** changes in the **Edit Object(s)** dialog.
Creating Instance Arrays

An array is a two-dimensional arrangement of objects, offset in the vertical and/or horizontal directions by specified amounts. A single instance is equivalent to a 1x1 array.

To create an array, select the instance and choose Edit > Edit Object(s), press Ctrl+E, double-click the MOVE/EDIT mouse button, or click the edit object(s) button.

In the Edit Object dialog, select the Instances tab.

In the Array parameters area, enter the horizontal and vertical repeat count and the X and Y spacing (Delta) between array elements.

You can also create an array by grouping instances (see “Grouping and Ungrouping Objects” on page 315) or by duplicating objects (see “Duplicating Objects” on page 328.)

Creating Arrays using the Mouse

You can quickly create an array of any instance by selecting an edge or corner and using the Move operation. The newly-created array will be an instance itself.
Simply select the edge of an instance and drag the mouse to create an array in one direction or select a corner to create an array in both the x- and y-directions, as illustrated below.

You can use **Draw > Ungroup** to convert the array to its component instances.

**Flattening an Instance**

**Cell > Flatten** removes the hierarchy of a cell. This is done by replacing all instances with the objects from the cell that they reference.

To flatten an instance by only one level, select it and use **Draw > Ungroup**. This command can be undone.
Assigning Names to Multiple Instances

You can quickly assign unique names to individual instances throughout the hierarchy using the `Cell > Assign Instance Names` command. Instance names are assigned using contents of the selected prefix field, plus the Separator (if any), plus the Start at value.

**Assign Instance Names**

**Assign names to**

- Selected instances in this cell (Cells)
- Instances in this cell (Cells)
- Instances in the cell (Cells) and hierarchy
- Instances in all cells

**Rename to**

- Custom prefix:
- Use cell name as prefix:

**Start at:**

**Separator:**

**Apply To**

- Only unnamed instances and conflicting instances
- All instances

**Exclude instances of cells:**

**Include instances of cells:**

**Assign names to**

Use these buttons to set the range of the name changes:

- **Selected instances in this cell**—renames just the instances in the active cell that are selected.
- **Instances in this cell**—renames all instances in the current cell.
- **Instances in this cell and hierarchy**—renames all instances in the current cell and each of the instances up the hierarchy to the parent cells.
- **Instances in all cells**—renames cell instances in the entire design.

**Custom prefix**

When checked, uses the characters in the entry field as the first part of the instance name. You may enter characters of any type or leave this field blank.

**Use cell name as prefix**

When checked, uses the instance’s master cell name as a prefix for the new name.
Assigning names to instances in all cells.
Prefix: TSM65
Separator: -
Start index: 99
Apply to: All instances
Exclude instances of cells:
(empty)
Include instances of cells:
(empty)
Ce1l Ce110
Ce11 Ce111
Ce11 Ce112
Ce112/U52 -> TSM65-99
Ce11 Ce113
Ce113/U53 -> TSM65-99
Ce11 Ce114
Ce114/cel17-C -> TSM65-99
Ce114/cel16-B -> TSM65-100
Ce114/cel16-A -> TSM65-101
Ce114/cel15-A -> TSM65-102
Ce114/cel11-A -> TSM65-103
Ce11 Ce115
Ce11 Ce116
Total renamed: 7
Setting the Scope of Action for Multiple Cell Operations

Operations on multiple cells or instances in L-Edit allow you to choose where your changes will be applied, from just the selected object to all objects in all open libraries in the design. Scope selection is used in many L-Edit commands, although the exact set of choices in the scope will vary according to what is applicable for the command.

OpenAccess databases require loading of all cells in the specified scope, which can be a time consuming operation. Carefully selecting the scope can reduce this load time.

- **Selection** — performs the operation only on objects that were selected in the active cell before the operation was initiated.
- **View** — performs the operation only on the active view of the cell.
- **Cell** — performs the operation on all view types of the active cell.
- **Hierarchy** — performs the operation on all instances instanced in (contained within) the active cell.
- **Library** — performs the operation on all instances in the library indicated.
- **All Libraries** — performs the operation on the target instances in all libraries in a design.
- **Libraries...** — performs the operation on the target instances in selected libraries in a design.
If **Libraries...** is selected, then a dialog is displayed to select the libraries to which the operation will be applied. The (**...**) button will also invoke this dialog directly.

The following commands use the multiple cell scope of action menu:

- Draw > Convert > Change Layers
- Draw > Convert > Snap to manufacturing grid
- Draw > Convert > Fracture Polygons
- Draw > Clear Persistent Rulers
- Draw > Assign GDS Data Types
- Cell > Replace Instance
- Libraries navigator (right-click menu) > Replace Instances > Of This Cell
- Libraries navigator (right-click menu) > Replace Instances > In Selected Cells
- Tools > Clear Generated Layers
- Tools > Clear Error Layer
- Tools > Add-Ins > Remove GDS Property Labels
- Cell Navigator > Status
- Verification Navigator > (command menu) > Delete Results

**Replacing Instances**

**Cell > Replace Instance** lets you replace instances of one or more cells with another.

L-Edit provides separate and independent controls for replacing cell instances. You can independently set which instances will be replaced; the extent of where the replacements will take place (from within a single cell to in all libraries); and what the instance(s) will be replaced with.
Note: Note that the replace instance operation cannot be undone.

Replace Instances

WHERE
- replace the instances found here

WHAT TO REPLACE
- replace these instances

REPLACE WITH THIS
- replace with this instance

Replace instances found in
This pull-down menu lets you set the range, from one cell to all libraries, where the replacement will be performed. Please refer to “Setting the Scope of Action for Multiple Cell Operations” on page 260 for a discussion of the options.

Instances to replace
Master of selected instances - replaces instances of the same master as the selected instances.
Instances of - replaces just the instances of the cell you select that have already been placed.

With instances of cell
Select the cell the instance will be replaced with.

Replace instance with respect to cell
Origin - aligns the origin of the replacement instance with the origin of the original instance.
Abutment - If the original selected instance has ports (L-Edit considers geometry on the Icon/Outline layer to indicate the presence of a port), this option will cause L-Edit to attempt to align the MBB of the new geometry with that of the original instance. Please see “Aligning Instances by Abut Ports,” below for more detail.

Aligning Instances by Abut Ports

When you select an instance and replace it, you have the option of aligning the selected instance and the replacement instance according to their origins or abut ports. When you choose the Abutment option, L-Edit will attempt to align ports on the replacement instances with ports on the original instance.
An **abut port** is a box port with text that matches the **Abutment** field in the dialog **SPR Core Setup—General**. When you replace an instance and specify alignment by abut port, L-Edit examines the selected instance and the instance you are replacing it with to see if their abut ports match. The check of abut port names is case-sensitive.

If the abut ports in the two cells have matching names and dimensions, L-Edit places the new instance in exactly the same position as the previous one. If the abut ports in the two cells do not match, or if there are no abut ports in either cell, L-Edit will prompt you with the following dialog for permission to align on the instances on their centers.

![I-Edit Warning](image)

**Replace Instances of This Cell**

As a shortcut, you can right-click in the cell list to replace instances.

When you use **Replace Instances > Of This Cell**, instances of the cell in the list that the pointer is over will be replaced with the cell that is active. In the example below, instances of “Channel_1,” which was highlighted in the cell list, will be replaced by “Channel_2” within “Cell0,” the cell that is active in layout.

![Replace Instances](image)

**Replace Instances in Selected Cells**

As a shortcut, you can right-click in the cell list to replace instances in the cell(s) that are selected. You may choose multiple cells for this operation.
When you use **Replace Instances > In This Cell**, instances in the highlighted cell will be replaced by the cell you specify. In the example below, instances of “csmin” will be replaced by instances of “nmos3p3v” within the six cells highlighted in the cell list.

![Replace Instances In dialog box](image)

- **Displays the cell that will be replaced.**
- **Select the cell the instance will be replaced with.**

### Retargeting the Library of Selected Cells

You can replace instances of cells from one library with instances of like named cells from a different library using the **Replace Instances > Retarget Library** command. First select the cells in the library navigator that you wish to replace, then invoke the Retarget Library command and specify the scope in which you want to replace instances, and the new library where the new cells are located. Cell-views in
the specified scope will be replaced with cells with the same cell and view name from the new library.

Replacing Multiple Instances using Edit Object(s)

You can also replace multiple selected instances, whether the same or different, with another instance using the Edit Object(s) dialog. When multiple instance are selected, the Instance of cell and Instance name fields will be disabled.

However, when you click on the Change button, L-Edit will open the Select Cell to Instance dialog. Use this cell list to pick the cell that will replace those selected in layout (see “Replacing Instances” on page 261). Note that any self-referential cells in the list will be disabled.

Note also that when you click OK you will return to the Edit Object(s) dialog, but the Instance of cell and Instance name fields will still be blank – they will not display or confirm the changes you have initiated.

Locking and Unlocking Cells Hierarchically

Hierarchical locking and unlocking allows you to lock and unlock cells and all of their instanced cells so that a locked cell cannot be modified during an edit in-place operation.
In hierarchical unlocking, the top-level cell is unlocked and L-Edit prompts you for permission to individually unlock each cell that is instanced in it. The dialog also gives the option to unlock all instanced cells.

Locking and unlocking will always be exclusive. For example, if cell A and cell B both instance cell C, and hierarchical lock is performed on cells A and B, all three cells will be locked. If cell A is subsequently hierarchically unlocked, cell C will be unlocked, including its instance in cell B.

Depending on the state of the selected cell and the associated file, some menu items will be disabled:

- If file is locked: All items except for Open, Info are disabled.
- If selected cell is instanced in another cell: Delete, Hierarchical Delete are disabled.
- If selected cell is locked: Flatten, Rename, Delete, Hierarchical Delete are disabled.

**Specifying the Fabrication Cell**

Before fabricating your design, you must supply your fabricator with the name of the cell that represents the top level of your design. If you do not specify this information, your fabricator may incorrectly identify this cell.

Identifying the fabrication cell instructs L-Edit to tag the cell as such when it exports a CIF file. The identified cell becomes the only top-level cell in the CIF file. (This feature is only available for CIF files. The GDS II format does not contain top-level cell information.)

Once a fabrication cell has been chosen, it will remain the fabrication cell until a new one is chosen, even if it ceases to be the top-level cell in your design. *Be sure to identify the fabrication cell before writing a CIF file!*
Identify the cell to fabricate by choosing **Cell > Fabricate.**

![Select Cell to Fabricate](image)

**File**

Name of the active TDB file (default). All open TDB files are listed in the pull-down menu.

Highlight a cell in the scrollable list and click **OK** to select it.

### Finding I/O Pads in the Fabrication Cell

The **Tools > Add-Ins > I/O Pad Crossreference** is a feature that can automatically generate the pad coordinates for probe cards or bonding diagrams that are generally required by the packaging houses. It analyzes the layout of the currently specified fabrication cell, and finds all I/O ports (pads) in that cell. Pads are recognized as objects on the user-specified layer (typically Overglass).

The output table includes the pad height under the column head “H” and width under the column head “W.”

![I/O Pad Cross-Reference](image)

Besides counting and reporting the number of pads, this command has two options:

- **Pad Overlay cell** Check this option to generate an overlay cell. An overlay cell is a new cell (the user must specify the name of this cell), containing box objects on the I/O pad recognition layer. Also, if a port is found within the pad box, on any layer, on any level of hierarchy, that name becomes associated with that pad. The overlay cell contains ports with this same name, assigned sizes and text directions that produce a visually appealing layout. This pad overlay cell is useful for communicating with package houses / bonding houses.
- **Cross-Reference file** The second option is to create a Cross Reference disk file containing the pad list information. This file can be in either of two formats, determined by the file extension specified. `.txt` files are simply tab-delimited lines containing the pad position, the pad name, and the x- and y- locations of the center of each pad. `.aif` files are in AIF format, a text format promoted by packaging houses (specifically Amkor); in addition to pad names and location, the AIF file contains the dimensions of the pad. If the AIF format is selected then the coordinates are relative to the middle of the cell's MBB, but if the TXT file is selected then the coordinate space of the cell (i.e. the cell origin) is used.
Wafer Tools

L-Edit Wafer Tool provides the ability to label the die on a wafer, to maximize the number of die on a wafer, to sort a group of die by their relationship to the wafer boundary and to trim process layers at the wafer boundary.

Populating the Wafer

The first step is to place the die on the wafer, using Tools > Wafer Tools > Place Die. This command adds as many die as it can within the wafer. No die will be placed inside objects on the keep-out layer or inside the bounding box of keep-out cells.

**Wafer**

Enter the size of the wafer in the **Diameter** field.

Select a **Boundary layer** for objects describing the wafer. Dies are placed only if they are completely inside objects on the wafer boundary layer.

**Die**

- **Die cell** - Select the cell that will be used to populate the wafer.
- **Die outline layer** - Objects on this layer define the die boundary. (The die boundary is calculated as the MBB of all objects on the SAR layer.)
- **Die rotation** - Rotation of the die being placed (0, 90, 180, 270).
- **Die separation** - Distance between die being placed. This is used to add space for slicing the wafer.

---

**Populate Wafer Dialog**

- **Wafer**
  - **Diameter**: 143857.143
  - **Boundary layer**: Active

- **Die**
  - **Cell**: 
  - **Outline layer**: Active
  - **Rotation**: 0
  - **Separation**: 0.000

- **Keepout**
  - **Cell**: 
  - **Layer**: 

- **Die termination**
  - **Vertical termination**: 
  - **Horizontal termination**: 

- **Die grouping**
  - **No grouping (ABAB)**
  - **Doublet grouping (AABB)**

OK | Cancel
**Die origin**

You can enter an x, y value for a **Fixed** origin.

If you select **Random**, you must also enter a value for the **Number of Iterations**. L-Edit will use that number of iterations to try placing die, with each iteration starting at a different random location. The iteration that places the most die inside the wafer will be used.

**Keepout**

Choose a **Keepout cell** so that die cells will not be placed within the MBB of keep-out cells and/or choose a **Keepout layer** so die cells will not be placed within objects on the keepout layer.

**Die termination**

Check **Vertical termination** to place termination cells at the beginning and end of each column and/or check **Horizontal termination** to place termination cells at the beginning and end of each row. Select a **Terminator cell** to specify the cell to be used at the ends.

**Die grouping**

Choose either **Doublet grouping (AABB)** to mirror every other die in each row or column, or **No grouping (ABAB)** to alternate.

Here is an example of a possible result:
The log file will indicate the starting location of the die array that is used to fill the wafer. All die not completely inside the wafer and not completely outside the keep-outs will be removed from the array. The starting location of the array is snapped to the manufacturing grid.

![Log File Screenshot](image-url)
Labeling the Die

The next step is to give each die in the cell a unique ID using OCR characters with **Tools > Wafer**
**Tools > Label Die**. Make sure to have Setup Application > UPI > “Update display” checked when you
run the die labeler.
Chapter 7: Cell Hierarchy and Views

Specifying the Fabrication Cell

The die labeler will place a sequence ID number in all die that have a CLOWN reference box. The dies are then sorted into rows and columns. Those die or locations on the wafer without a CLOWN reference box will not get an ID but will be used when sequencing the ID numbers.

**Label die by column & row**

(0101, 0102, 0103, ...)

This is a radio button - choose either this option or “Label die sequentially.”

Labels each die with its column and row number. Dies are sorted geometrically into rows and columns so that die that align horizontally will be in the same row and those that align vertically will be in the same column. If a die is missing in the middle of a row or column, it will be counted but not labeled. For example,

(0101, 0102, ...)

**Column label /Row Label**

*Label Format* displays the column or row number in either hexadecimal, decimal, octal, alphabetic (A, B, C, ... AA, AB, AC, ...), or alphanumeric (0, 1, 2, ..., 8, 9, A, B, C, ..., 00, 01, 02, ..., 0Y, 0Z, 10, ...) format

*Starting label* indicates the number or letter that will be used for the first column or row.

*Label Width* - Enter the number of characters for the width of a column or row label. Column numbers that are shorter than the width will have zeros prepended to fill the width. Letters that are shorter than the width will have spaces added. For example, if the width is 2, then it will be 00, 01, 02, ...; if the width is 3, then it will be 000, 001, 002, ...).

**ID Format:**

Sets the format for the die label as either Row#Row#Col#Col# (RRCC) or the reverse Col#Col#Row#Row# (CCRR).

- RowColumn - sorts by row first, then column
- ColumnRow - sorts by column first, then row

**Label die sequentially**

(1, 2, 3, ...)

This is a radio button - choose either this option or “Label die by column & row.”

Labels each die sequentially based on the label format. Die are labeled sequentially across the row.

- Select a Label format from the pull-down list: Hexadecimal, Decimal, Octal, Alphabetic, or Alphanumeric.
- Set the number of characters in Label width. IDs that are shorter than the width will have zero added to the beginning of them.
- Starting label — use this to set the value of the first ID.
- Increment amount — this is the amount to increment each label (i.e. an increment of 2 would result in 1, 3, 5, 7, ... or A, C, E, G, ... or similar)
- Zig-zag labels — This will label the even rows left to right and odd rows right to left (sometimes called a “serpentine” pattern.)

**Die MBB**

Definitions:

- **MBB** — a Minimum Bounding Box is the smallest rectangle that contains all of the objects on a specific layer or in a cell.
- **CLOWN** — the chip location on wafer number is a box on a special layer that designates where to place the die label.
- **SAR layer** — is a Set and Repeat Layer containing objects that define the die outline.
Check **Use MBB of the cell that contains the CLOWN object as the die's MBB** to use the MBB of the cell that contains the CLOWN object as the die's MBB. If this is checked, you do not need to set the SAR layer.

Wafer Tools keeps track of the location and orientation of the CLOWN object at the top level and places the ID based on this box. For example, if the CLOWN box is in a cell that is instanced and mirrored in the die cell, the die label will be mirrored also.

WaferTools can place labels horizontally or vertically as defined by the orientation of the CLOWN box in the die cell.

Alternately, you can check **Die Outline (SAR) layer** to choose a layer that Wafer Tools will read for objects that defines the die outline.

Wafer Tools uses the MBB of all objects on the SAR layer to determine the size of the die. The size of the die determines which row and column the die is in. All die that have their center inbetween the lowest die in the column will be in that column.

When a SAR layer is encountered, the chip labeler will stop looking for a SAR object down the hierarchy but will continue looking for a CLOWN object. (The CLOWN box has to be at the same level or lower level of the hierarchy that the SAR objects are located on.)

**Add frontside IDs**

Check this option to place the die label on the wafer front, and select a Frontside label placement (CLOWN) reference layer.

You must also select a Font layer on which the ID number will be placed, and a Font size that is either Fixed or Variable.

The Variable option uses the size of the box on the label placement (CLOWN) reference layer to set the height of the letters. It will use the short dimension of height and width of the box to determine the size of the ID.

**Add backside IDs**

(same as Frontside ID fields with this additional option:)

Check **Mirror ID** to mirror every other die in each row or column.

**Start labeling from:**

Choose where labeling will start, from
Lower Left - start to number die from the upper left corner then across and down.
Upper Left - start to number die from the lower left corner then across and down.
Lower Right - start to number die from the lower right corner then across and down
Upper Right - start to number die from the upper right corner then across and down

**Force ID label to face left to right or bottom to top**

This forces the ID label to face left to right or bottom to top. If this is unchecked, the chip labeler uses the CLOWN object and its transformation to the top level to calculate how the label will face. This option is useful if you have some of the die mirrored.

**Place labels in cell:**

The cell in which to place all die ID labels.
Snap ID labels to manufacturing grid

The ID number will be formatted as specified in the Die Labeler dialog and will be done using an OCR-A font, as shown below.

Die are labeled in Row-Column format:

If the ID of any die is larger than its CLOWN box, L-Edit issues a warning to the log file:

Chip Labeler Version 16.01
L-Edit Version: Version 16.01
File: Wafer.tdb
Top cell: Wafer
May 27, 2013 06:24:39 PM
Number of Dies: 4369
Number of Rows: 78
Number of Columns: 78
**Mapping the Die**

The third step is to map the die, using **Tools > Wafer Tools > Map Die**.

![Die Map Dialog](Image)

**Wafer**
- Die
- Bad die
- Die marker
- Die handling

(To be completed in version 16.03.)
# Drawing Objects

## Object Types

The basic task in designing layout is drawing objects, which represent the elements and patterns of the circuitry to be fabricated.

There are several types of objects you can draw. Each object type is associated with a tool on the Drawing toolbar which you use to draw the corresponding object.

<table>
<thead>
<tr>
<th>Object type</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box</td>
<td><img src="image" alt="Box Icon" /></td>
<td>A shape characterized by four 90° corners.</td>
</tr>
<tr>
<td>Polygon</td>
<td><img src="image" alt="Polygon Icon" /></td>
<td>A shape characterized by an arbitrary number of vertices connected by straight edges to form a closed (possibly self-intersecting) figure.</td>
</tr>
<tr>
<td>Wire</td>
<td><img src="image" alt="Wire Icon" /></td>
<td>A shape consisting of one or more rectangular segments, of equal width, joined at common ends.</td>
</tr>
<tr>
<td>Circle</td>
<td><img src="image" alt="Circle Icon" /></td>
<td>A shape characterized by a center (point) and a radius.</td>
</tr>
<tr>
<td>Pie Wedge</td>
<td><img src="image" alt="Pie Wedge Icon" /></td>
<td>A section of a circle characterized by a center, a radius, and a sweep angle.</td>
</tr>
<tr>
<td>Torus</td>
<td><img src="image" alt="Torus Icon" /></td>
<td>A section of a circle characterized by a center, two radii (inner and outer), and a sweep angle.</td>
</tr>
<tr>
<td>Port</td>
<td><img src="image" alt="Port Icon" /></td>
<td>A shape and associated label used for electrical connection.</td>
</tr>
<tr>
<td>Ruler</td>
<td><img src="image" alt="Ruler Icon" /></td>
<td>A line with a choice of end styles and optional tick marks, used to measure layout.</td>
</tr>
<tr>
<td>Instance</td>
<td><img src="image" alt="Instance Icon" /></td>
<td>A symbolic representation of a cell at a specific location and orientation in another cell. For information on how to create an instance, see “Creating Instances” on page 251.</td>
</tr>
<tr>
<td>Via</td>
<td><img src="image" alt="Via Icon" /></td>
<td>An electrical connection between layers that goes through one or more adjacent layers. For information see “Generating Via Cells” on page 391.</td>
</tr>
</tbody>
</table>
Selecting a Layer

Before you draw an object, you must select a layer. When a layer is selected, the layer icon in the Compact Layer Palette is outlined, and the name of the layer appears at the top of the Layer Palette. Any objects you create during a draw operation will be on the selected layer and will display the color and pattern specified for that layer.

You can select a layer in three ways:

- Click an icon or pick a layer from the drop-down list in the Compact Layer Palette.
- Select the desired layer from the list in the Layer palette.
- Choose Draw > Pick Layer (shortcut key A) to change the current layer to the layer of the object which the cursor is over, whether instanced geometry or not and regardless of any object selected.

After you specify a layer, drawing an object involves two basic steps: (1) selecting a drawing tool and (2) executing a drawing operation.

Selecting a Drawing Tool

To select a drawing tool, click on a button in the Drawing toolbar. You will remain in the same drawing mode until you select another tool.
Chapter 8: Drawing Objects

Selecting Angle Constraints for Drawing Tools

You can limit the range of tools displayed in the Drawing toolbar to match your design, either by right-clicking in the Drawing toolbar, or using the Drawing mode field in Setup > Application—General to choose Orthogonal, 45 Degrees, All Angle or All Angle & Curves.

- Press and hold the **Ctrl** key to constrain drawing and editing to 45-degree movement.
- Press and hold the **Shift** key to constrain drawing and editing to 90-degree movement.
- Press and hold **Ctrl + Shift** to allow all-angle drawing and editing.

Changing Angle Constraints while Drawing or Editing

You can use hot keys to temporarily change your drawing or editing mode.

- Press and hold the **Ctrl** key to constrain drawing and editing to 45-degree movement.
- Press and hold the **Shift** key to constrain drawing and editing to 90-degree movement.
- Press and hold **Ctrl + Shift** to allow all-angle drawing and editing.

Drawing in Outline Mode

While you are drawing or editing an object you can toggle rendering of that object from filled mode to a transparent outline-only mode so that objects below remain visible. Use the **Tab** key or **Ctrl+I** to perform this toggle. (See “Displaying Instance Insides While Drawing and Editing” on page 203 for more information.)

Drawing Objects

The starting point for drawing any object is its anchor point. To draw an object, select a drawing tool and position the crosshair pointer where you want the anchor point to be. Click the DRAW (left) mouse button to begin drawing the object.

Note that in order to draw circles or other curves, you must set the Drawing toolbar to All Angle & Curves.

While you are drawing or editing an object you can toggle rendering of that object from filled mode to a transparent outline-only mode so that the objects below remain visible. Use the **Tab** key (or **Ctrl+I**) to perform this toggle. (See “Displaying Instance Insides While Drawing and Editing” on page 203 for more information.)
Note that you cannot draw on a layer that is locked.

**Boxes**

The anchor point is one of the corners of the box.

Hold the DRAW mouse button and drag the pointer away from the anchor point to determine the opposite corner (and therefore the length and width) of the box. Release the DRAW button at the desired opposite corner.

To edit boxes using a non-graphic interface, see “Edit Object(s) for Boxes” on page 354.

**Circles**

The anchor point is the center of the circle.

Hold the DRAW mouse button and drag the pointer away from the anchor point to determine the radius of the circle. Release the DRAW button at the desired radius.

To edit circles using a non-graphic interface, see “Edit Object(s) for Circles” on page 357.

**Pie Wedges**

The anchor point is the center of the pie wedge. The mouse buttons become VERTEX, BACKUP, and END, respectively.

To create a pie wedge, click the VERTEX (left) mouse button at the anchor point and drag the pointer away from the anchor point to determine the radius of the pie wedge (indicated by a thin line). Click or release the VERTEX mouse button at the desired radius. Drag the pointer again to determine the end angle of the pie wedge. The angle is always calculated counterclockwise. Click the VERTEX or END (right) mouse button to complete the pie wedge. Click the BACKUP mouse button to reverse each step before the pie wedge is completed.

To edit pie wedges using a non-graphic interface, see “Edit Object(s) for Pie Wedges” on page 358.

**Tori**

The anchor point is the center of the torus. The mouse buttons become VERTEX, BACKUP, and END respectively.

To create a torus, click the VERTEX (left) mouse button at the anchor point and drag the pointer away from the anchor point to determine the first radius of the torus (indicated by a thin line). Click or release the VERTEX mouse button at the desired first radius. Drag the pointer again to determine the sweep angle and the second radius of the torus. Click the VERTEX or END (right) mouse button to complete the torus. Click the BACKUP mouse button to reverse each step before the torus is completed.

To edit tori using a non-graphic interface, see “Edit Object(s) for Tori” on page 359.

**Polygons and Wires**

The anchor point is the first vertex of the polygon or wire. Polygons and wires can have any number of vertices. The mouse buttons become VERTEX, BACKUP, and END, respectively.
To create a polygon or wire, click the VERTEX (left) mouse button at the anchor point and drag the pointer away from the anchor point to determine the second vertex. Repeat the process for each successive vertex. Click the BACKUP mouse button to remove the last vertex that was placed.

Click the END mouse button at the last vertex to complete the object. When you click the END button, coincident vertices (two or more vertices occupying the same location) and colinear vertices (three or more vertices lying on the same straight line) are eliminated.

**Warning:**
The appearance of a wire on the screen does not guarantee it will form a connection with an object when the chip is fabricated. Check with your manufacturer regarding the type of join and end styles to use in your design.

To edit polygons and wires using a non-graphic interface, see “Edit Object(s) for Polygons” on page 355 and “Edit Object(s) for Wires” on page 356.

**Self-Intersecting Polygons and Wires**

Two common design errors involve self-intersecting polygons and polygons with ambiguous fills. Either can be misinterpreted by the manufacturer and result in an incorrect object mask.

L-Edit will display a warning when drawing or editing a polygon if the polygon intersects itself at any point. The **Fix** option breaks up the polygon into multiple polygons at its self-intersecting points. A similar warning is displayed for self-intersecting wires. You can turn off these and other warnings with **Setup > Application—Warnings**.
Self-Intersecting Polygons

An example of a self-intersecting polygon is shown below:

Ambiguous Fill Polygons

Certain types of self-intersecting polygons result in an ambiguous definition of the filled area of the polygon. An example of a polygon with an ambiguous fill is shown in the following illustration. Depending upon the manufacturer’s convention, the white enclosed box might or might not be filled when fabricated.
Similarly, in the figure below the desired fill in the region of intersection cannot be determined unambiguously.

![Ambiguous Fill Diagram](image)

**Winding Number**

The areas of a self intersecting polygon can be classified by a winding number. In the illustration below, each value represents the *winding number*—the number of times that a point in the polygon is circumscribed when the figure is traced in one direction around its perimeter.

![Winding Number Illustration](image)

L-Edit interprets an area with winding number equal to zero as unfilled, an area with an odd winding number as filled, and an area with an even number as unfilled. Polygons with a winding number greater than or equal to two are identified as ambiguous polygons, since other CAD systems may interpret the filled area differently.

**Curves**

Using an all-angle tool and the ARC mouse shortcut, you can convert an edge of an existing orthogonal, 45-degree, or all-angle polygon to a curved edge (the portion of a circular arc with endpoints at the two vertices of the original edge). Boxes cannot be converted to curves.
This operation is only possible on existing polygons—you cannot directly draw a polygon with curved edges. (See “Chamfers and Fillets” on page 286 for converting the shape of vertices.)

**Converting a Straight Polygon Edge to a Curve**

- Select the **All Angle & Curves** display mode.
- Using the SEL EDGES mouse button (Ctrl+LEFT), click to select a single edge of the polygon.

**Note:** To convert an edge to a curve, you must select only the edge of an object, not the entire object. If you select the entire object, it is only possible to add a vertex to any of its edges.

- Using the ARC mouse button (Ctrl+MIDDLE or Alt+Ctrl+LEFT), drag the cursor to increase or decrease the curvature of the arc.

Each curve is defined by a specific *curve height*, illustrated in the following section.
Curve Height

Curve height is the perpendicular distance, in display units, between the chord that connects the two endpoints of the curve and the midpoint of the curve. The curve height will be positive or negative depending on the order in which the vertices of the polygon were created.

Chamfers and Fillets

A chamfer is a beveled edge connecting two surfaces. If the surfaces are at right angles, the chamfer will typically be symmetrical at 45-degrees. Similarly, a fillet is a curved connection between two surfaces, which is concave for an interior corner and convex for an exterior corner as shown below.
Chamfers and fillets can be applied to polygons and wires in L-Edit. First select one or more objects, then use **Draw > CurveTools > Chamfer...** or **Draw > CurveTools > Fillet...** to open the **Chamfer/Fillet** dialog shown below.

**Chamfer/Fillet**

- **Operation**: Select **Chamfer** (beveled) or **Fillet** (curved)
- **On**
  - **Selected vertices**: Choose to perform the operation on only selected single vertices of the object. If you use this option, you click **Add** to return to the layout, where you can click on individual vertices to select them or **Remove** to deselect them from the operation.
  - **Entire object**: Choose to chamfer or fillet all vertices on the object.
- **Which vertices**
  - **Convex polygon vertices**
  - **Concave polygon vertices**
  - **Wire vertices**
  - **Distance**: Enter the value in Microns
  - **Radius**: Enter the value in Microns
- **Display markers at vertices that can't be chamfered or filleted**
- **Create log file**

### Operation

Select **Chamfer** (beveled) or **Fillet** (curved)

### On

Choose **Selected vertices** to perform the operation on only selected single vertices of the object. If you use this option, you click **Add** to return to the layout, where you can click on individual vertices to select them or **Remove** to deselect them from the operation.

Choose **Entire Object** to chamfer or fillet all vertices on the object.

### Which vertices

Use these options to limit the vertices on which the operation is performed. You can choose to limit by position, using **Convex vertices** or **Concave vertices** for polygons and by **Distance**, which is essentially the size of the fillet or chamfer. See “Setting the Chamfer/Fillet Distance or Radius,” below, for a complete discussion of this value, and an illustration of how wire ends are preserved.

To fillet or chamfer wires, check **Wire Vertices** and enter the desired **Radius** value. Note that wire width and end style are both preserved, and that the chamfer/fillet length is applied to the wire centerline.

### Display markers at vertices that can't be chamfered or filleted

When checked, L-Edit places a marker at the vertices on which it is unable to perform the operation (usually because the line segment is shorter than the Distance or Radius value). These markers are heavy black circles.

### Create log file

When checked, L-Edit creates a log file that opens when the operation is complete.

### Setting the Chamfer/Fillet Distance or Radius

The radius of a fillet or the cut of a chamfer is determined by the intersection, at two points, of the polygon and a circle established by the **Distance** value. On the polygon below, the **Distance** value...
along each edge of a vertex (A) intersects with the diameter of a circle having a radius of the same **Distance** value (B) to create the fillet radius.

Note that the edge length for each vertex must be more than double the radius selected for that vertex to be filleted. If the vertex is shorter than twice the radius there could be some overlap. Selecting a vertex has the same restriction.

The red circle illustrates how a concave fillet is set by the distance value. The gray circle illustrates how a convex fillet is set by the distance value.

A wire and polygon before (left) and after (below) the chamfer operation. Note that the wire end style and width are maintained.

For wires, the radius value is applied to the centerline only, resulting in differing lengths on the inside and outside vertices.

For polygons, the distance value is applied consistently to each vertex.
Electrical Ports

Prior to version 16.00, ports in L-Edit were text and a shape combined into a single object, with the shape being a point, line, or rectangle. With version 16, L-Edit supports true electrical ports, where the shape and label are distinct objects, and a single port may contain more than one shape. (Electrical ports in L-Edit are analogous to “pins” in other tools.)

An electrical port must have a designated terminal name, and it must also be drawn on a layer-purpose pair – you cannot draw an electrical port on a layer without a purpose. (See “Legacy Ports” on page 291.) It is not required to contain a label.

All shapes on the same port are by default “strongly connected,” so that you can connect to any one of them.

Drawing and Editing Ports

To draw a port, select a layer with an attached purpose and choose one of the port buttons. Port shapes can be rectangular, orthogonal, 45-degree or all-angle.
When you complete drawing the port shape, L-Edit will open the **New Port** dialog:

**Terminal name**
- Enter the name for the collection of ports and pins this port will belong to.

**Text size**
- Enter a size for the text label.

**Layer**
- If you want to change the layer for a port shape, use this pull-down menu.

**Terminal type**
- (This attribute is not currently supported by L-Edit, but the setting is retained and editable in the OpenAccess database to integrate with tools that use these settings.)

**Port access direction**
- Check these boxes to allow port access for routing at the **Top**, **Bottom**, **Left** or **Right**. (This attribute is not currently supported by L-Edit, but the setting is retained and editable in the OpenAccess database to integrate with tools that use these settings.)

**Text position**
- Click on one of these boxes to position the anchor of the port label, and select either **Horizontal** or **Vertical** for the text direction.

**Coordinates**
- Use these optional fields to position the anchor point of a port shape by entering absolute coordinates.

**Preview**
- This area displays how port text will be positioned.
Adding Additional Port Geometry

An L-Edit port can have any number of shapes that are connected as long as they have the same terminal name.

Once you have drawn the first port shape, you can “push in” to edit it or to add other shapes. “Pushing in” is comparable to editing-in-place - it allows you to edit the individual objects and label that comprise the port as a whole.

To push in to a port, first select the port in the layout, then choose a layer and a port drawing tool, and press Page Down. This will place you in port editing mode, where any geometry you draw.

L-Edit will indicate that you are in port editing mode by displaying “EDITING” in front of the list of objects that belong to the port, as shown below.

![Port AA+ selected](image1)

Port AA+ selected

![“Pushed in” to port AA for editing](image2)

“Pushed in” to port AA for editing

Terminals and Pins

A terminal is a port or net connection for a block, such as In, Vdd, or Gnd. Each terminal has a unique name and is considered the connection to the next level up in the hierarchy. Each terminal can have multiple pins, which may be strongly or weakly connected. Each pin can also have multiple shapes or geometry. (Multiple shapes for pins are usually used to indicate that you can connect to the pin through more than one layer.)

You can edit the terminal name and other settings of a port by selecting the port and using the Ports tab in Edit > Objects (shortcut Ctrl+E). The Must connect field is a list of terminal names that must be connected.

Legacy Ports

Prior to version 16.00, ports in L-Edit were a single object containing text and a shape. These ports are referred to as “legacy ports.” Similarly, as of version 16.00, a layer with no purpose assigned is referred to as a “legacy layer.” In order to maintain compatibility with designs from previous versions of L-Edit, you can still create legacy ports, on legacy layers.

A legacy port can be a point, a line, or a two-dimensional box. The anchor point of a point port is the location of the port. The anchor point of a line or box port is a corner of the port.

To draw a point port, position the pointer at the anchor point and press the DRAW mouse button.

To draw a box port, click-and-drag the DRAW mouse button away from the anchor point to the desired length and width of the box. Release the DRAW button at the desired location of the opposite corner.

To draw a line port, click-and-drag the DRAW mouse button away from the anchor point in only the vertical or horizontal direction. Release the DRAW button at the desired location to end the line port.

When you release the DRAW mouse button, the Edit Object(s)—Ports dialog opens and prompts you for the Port name. At this point you can also modify other attributes of the port, including GDSII data type, text size, coordinates, text orientation, and text alignment.
For more information on editing ports using a non-graphic interface, see “Edit Object(s) for Ports” on page 360.

**Legacy Layers**

Beginning in version 16.00, a layer with no purpose assigned is considered a “legacy layer.” In order to maintain compatibility with designs from previous versions of L-Edit, legacy layers are used when legacy ports are drawn.

**Text Labels**

L-Edit supports *text labels*. A text label is a text string located at an anchor point. Unlike a port, there is no geometry associated with a text label. The text has a specified size, justification, and horizontal or vertical orientation with respect to its anchor.

To create a text label, click on the Text Label button “T” on the drawing toolbar, then click in the layout to pick the location for the anchor, which will launch the **New Label** dialog. The anchor is displayed as a small plus sign.
L-Edit stays in text label mode until you choose another editing mode.

You can use **Edit > Allow Label Selection** to set whether text labels can be selected and **Ctrl+E** to edit them.

- **Name**: Enter the label text.
- **Text size**: Enter the text size (in display units).
- **Layer**: Select the layer on which the text will be drawn.
- **Label**: Enter a value (in display units) of the X- and Y- location of the text anchor.
- **Text Justification**: Pick a direction (**Horizontal** or **Vertical**) for the text, and the alignment of the text with respect to the anchor (**Left**, **Right** or **Above** and **Right**, **Left** or **Center**).

### Rulers

Note that there are two kinds of rulers in L-Edit. The following section describes **persistent rulers**. There are also **temporary rulers**, which are a form of marker—see “Temporary Rulers” on page 333.)

The anchor point of a persistent ruler is one of the endpoints of the ruler. Hold the DRAW mouse button and drag the pointer away from the anchor point to determine the other endpoint (and therefore the length and orientation) of the ruler. Release the DRAW button at the desired endpoint.

You may draw a ruler on any layer, and you can choose an orthogonal, 45° or all-angle ruler using the toolbar icons.
**Editing and Deleting Rulers**

Ruler behavior is somewhat different than other objects. To select a ruler, you need to use a selection box around it. Once it is selected, you can edit a ruler (see below) or delete it.

If you use the **Delete** button, only the selected object(s) are deleted, and the operation can be undone. Alternately, you can use **Draw > Clear Persistent Rulers**. This allows you to select the scope of cells from which rulers are deleted—regardless of whether they are selected— but it cannot be undone.

---

**Edit Object(s) for Rulers**

Default ruler settings, including the layer on which rulers are drawn, text size, end style, and tick mark settings, are entered using **Setup Design—Drawing**. To change the attributes of a specific ruler, select it in layout and use **Edit Object(s)**.
**Size**
The letter height, in display units, for text associated with persistent rulers.

**Display:**
Persistent ruler text can be displayed in one of four ways. Select the desired option from the drop-down list.

**No text** - No values are visible.

![Centered Display]

**Centered** - The total length of the ruler is displayed at the center of the ruler.

![At End Points Display]

**At end points** - Values are displayed at the start and end points of the ruler.

![At Tick Marks Display]

**At tick marks** - Numbers are displayed at each major tick mark along the ruler (default display).

**Angle**
Sets the angle of the number display, calculated clockwise as the angle from the 270°.

The angle is measured clockwise from horizontal - in this example it is 35 degrees.

**Show text on the other side of the ruler**
Relocates numbers and tick marks from one side of the ruler to the other.

**Tick marks**
Specifies the distance between tick marks in the Major and Minor fields, respectively. Type the desired spacing in the appropriate field to set the distance between tick marks, in display units. Major tick marks are rendered twice as long as minor tick marks.

**Visible**
Check this box to display tick marks for the ruler.

**Symmetric**
When this box is checked, tick marks extend above and below the persistent ruler.

**End style**
Persistent ruler lines contain one of two end styles: arrows or tick marks. Highlight the illustration in the drop-down list to select the default end style.
Wire Styles

An L-Edit wire consists of one or more rectangular segments joined at common ends. All segments in the wire have the same width, but each segment can have a different length. The point where two segments meet is called a *join*. The *endpoints* of a wire are the two segment ends which are not involved in joins.

A wire is characterized by a *style*, consisting of three properties:

- *Width* (in display units—different from the “width” reported in the status bar when a wire is selected).
- *End style* (the appearance of the wire’s endpoints).
- *Join style* (the appearance of the wire’s joins).
When you select a wire, the width value reported in the status bar is the x-width of the minimum bounding box of the whole wire, as shown here:

![Minimum bounding box](image)

Width reported in status bar

### End Styles and Join Styles

L-Edit recognizes three end styles and four join styles. These styles affect the appearance of wires on the screen only, and changing a wire’s style does not affect its endpoint or vertex coordinates. Contact your fabricator to determine the actual method of fabricating wires and what end and join styles the fabricator supports.

**Warning:** It is critical to verify that your fabricator interprets wires in the same manner as your layout. Otherwise the actual chip fabricated may be very different from what you wanted.

End styles include:

<table>
<thead>
<tr>
<th>End styles</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butt</td>
<td>Flush with the endpoint.</td>
</tr>
<tr>
<td>Round</td>
<td>“Capped” with a half-circle whose diameter equals the wire width.</td>
</tr>
<tr>
<td>Extend</td>
<td>Extended past the endpoint for a distance equal to half the wire width.</td>
</tr>
</tbody>
</table>

Join styles include:

<table>
<thead>
<tr>
<th>Join styles</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layout</td>
<td>The adjoining segment ends are extended to a distance equal to half the wire width. The resulting gap is filled with a triangle. This is the default join style.</td>
</tr>
</tbody>
</table>

This join style corresponds most closely to the interpretation of wires used by most fabricators. We recommend using this join style exclusively in your designs. (Wires created in versions of L-Edit previous to version 5 are automatically converted to the join layout style.)
The figure below illustrates various end and join styles.

<table>
<thead>
<tr>
<th>End style</th>
<th>Join style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butt</td>
<td>Layout</td>
</tr>
<tr>
<td>Round</td>
<td>Round</td>
</tr>
<tr>
<td>Bevel</td>
<td>Bevel</td>
</tr>
<tr>
<td>Miter</td>
<td>Miter</td>
</tr>
</tbody>
</table>

Wire Style Defaults

When a wire is first created, its style is taken from the default setting for the layer on which the wire is drawn, specified by choosing `Setup > Layers`. You may change the wire style parameters in the Default wire setting area in the Setup Layers dialog.

Before you draw wires for the first time, or if you are setting up technology files for others who may use wires, set the wire defaults for each layer according to whether your likely output format will be CIF or GDS II.

For CIF, use wires with the `extend` end style and the `layout` join style.

For GDSII, allowable combinations of end and join styles are shown in the table below. All other combinations of end and join styles will produce an error message when exporting to a GDSII file.
After you draw an individual wire you can change its individual style with Edit > Edit Objects.

**Wire Utilities for Editing Wires**

L-Edit provides several utilities to make editing wires easier. Draw > Wire Utilities allows you to join two or more wires together into one wire, extend the end segment of several wires to a new location, or slice one wire into two wires.

**Note:** Note that although the Wire Join operation can be performed on wires of a different widths without generating an error message, wires having a different width cannot actually be joined.

**Adding a Wire Section**

You can insert new wire sections into an existing orthogonal or 45° wire. First select an orthogonal or 45° wire to enable the command. Use Draw > Add Wire Section to switch to Add Section mode, and click on the selected wire at the point where you want to add a new wire section. L-Edit automatically draws the new section on the same layer on which the existing wire is drawn. To return to drawing mode, use the CANCEL mouse button. Areas where the wire intersects itself will be displayed as white.

**Joining Wires**

The Join command connects two or more selected wires. Draw > Wire Utilities > Settings lets you set whether wires must be touching to be joined.

<table>
<thead>
<tr>
<th>End style</th>
<th>Join style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extend</td>
<td>Layout</td>
</tr>
</tbody>
</table>

Join wires only if they touch at their endpoints

When this option is checked, two wires will only be joined if their endpoints are touching. End style is ignored.
When the touch option is not checked, all wires in the selection list will be joined by creating a wire segment between each closest two endpoints.

As shown in the following illustration, a wire segment is inserted from endpoint to endpoint. Thus, if the two endpoints do not have a common x or y value, an all-angle wire segment will be added.
Extending Wires

Extend will extend the end segment of the selected wires to the point where you click. Use Draw > Wire Utilities > Extend to extend the selected wires. It will ask you to click where you want to extend the wires to. It will then extend the closest end segment of each wire to point where you clicked.

Extend will not change the angle of the end segment. For non-orthogonal end segments, it will extend them to the intersection point between the end segment and the line that is perpendicular to the end segment and goes through the clicked point.
Resume Drawing a Wire from its Endpoint

You can restart drawing a wire from its end point by pressing **Ctrl+middle mouse button** when the cursor is directly over the endpoint. Make sure the wire is deselected before you launch this operation.

![Diagram showing the process of resuming wire drawing](image)

L-Edit will resume drawing the wire, rather than just editing it.

![Diagram showing the process of resuming wire drawing](image)
**Slicing Wires**

Slice will break a single wire into two wires. Use **Draw > Wire Utilities > Slice** to slice a wire. It will ask you to click where you want to slice the wire. It will then slice the wire at the closest point on the wire to where you clicked.

![Click on wire where you wish to slice it.]

**Fixing Wires With Short Segments**

**Draw > Wire Utilities > Fix Wires with Short Segments** finds and corrects short segments on any wire that is shorter than half the wire width, by removing them and then projecting and connecting the remaining wire segments. The scope of operation choices are “selected,” “cell (view),” and the entire “library.” This is an all-angle function - changes are not confined to orthogonal geometry. It will honor locked layers.

After completing the operation L-Edit opens a simple log showing the date, time and the number of wire changed or deleted. This utility is used mainly to clean up DXF files.

**Note:** Changes made using this dialog cannot be reversed.
Object Snapping

During drawing, moving or editing, you can use object snapping to “snap” the vertices and other control parameters of objects to the manufacturing grid.

L-Edit will snap to locations down the hierarchy on unmerged geometry. However, L-Edit will not snap to hidden geometry or an instance with its inside hidden.
Snapping Existing Geometry to the Manufacturing Grid

You can use Draw > Convert > Snap to Manufacturing Grid to snap existing objects to the manufacturing grid, choosing all or just selected objects in a cell or file.

Snapping to the manufacturing grid is an effective way to remove vertices from polygons with very high resolution of vertices, while still maintaining the shape of the original polygon. The snapped polygon will not deviate from the original polygon by more than the snap grid. Alternately, polygons with too many vertices can be fractured into several smaller polygons with fewer vertices (see “Fracturing Polygons” on page 350 and also “Setup Design–Grid” on page 82.)

When you issue the “snap to manufacturing grid” command:

- Vertices of boxes, polygons, ports, and rulers are snapped to the manufacturing grid.
- Vertices on wire centerlines are snapped.
- Instance origins are snapped, and array delta values are snapped such that the origin of array elements are on the manufacturing grid.
- The curve height of curved segments of polygons are snapped to an integer multiple of the manufacturing grid, and the radius of pie wedges and tori are snapped to an integer multiple of the manufacturing grid.
- The center of a circle is snapped to the manufacturing grid, and the radius is snapped to an integer multiple of the manufacturing grid.
**Vertex Snapping**

Vertices of boxes, polygons, ports, and rulers are snapped to the manufacturing grid.

The small black square on the upper right corner of the blue rectangle shows that snapping is on, and in this case is set to vertex snapping.

**Intersection Snapping**

The small black cross indicates where intersection snapping will begin.

Newly drawn geometry snaps to the intersection points of the neighboring geometry.
Object Snap Setup

Object snap options are set in the Setup > Design—Object Snap dialog.

Enable Object Snapping - Enables or disables object snapping. When disabled, no object snapping will occur regardless of the individual snap state. (Shortcut keys Alt +S and also F9.)

Vertex - Vertex snapping - Snaps the mouse to vertices of boxes, polygons, wire centerlines, pie wedges, tori, and port boxes. During a drawing or editing command, this snap mode can be toggled by pressing Shift+V.

Midpoint - Midpoint snapping - Snaps the mouse to the midpoint of edges of boxes, polygons, wire centerlines, pie wedges, tori, and port boxes. During a drawing or editing command, this snap mode can be toggled by pressing Shift+D.

Edge - Edge snapping - Snaps the mouse to nearest point on the edge of boxes, polygons, wire edges, pie wedges, tori, and port boxes. During a drawing or editing command, this snap mode can be toggled by pressing Shift+E.

Intersection - Snaps the mouse to the intersection of the edges of boxes, polygons, wires and curved edges of polygons.
Center (or wire centerline)  **Center snapping** - Snaps the mouse to the center of circles, boxes, pie wedges, tori, wire centerlines and curved edges of polygons. During a drawing or editing command, this snap mode can be toggled by pressing **Shift+C**.

**Quadrant**  **Quadrant snapping** - Snaps the mouse to nearest quadrant on a curved edge of circles, pie wedges, tori, and curved polygons.

**Pin**  **Pin snapping** - Snaps the mouse to nearest point on the edge of boxes, polygons, wire centerlines, pie wedges, tori, and port boxes. During a drawing or editing command, this snap mode can be toggled by pressing **Shift+T**.

**Instance**  **Instance snapping** - Snaps the mouse to nearest point on the edge of boxes, polygons, wire centerlines, pie wedges, tori, and port boxes. During a drawing or editing command, this snap mode can be toggled by pressing **Shift+S**.

**Move cursor to snap point**  When this option is enabled, when the cursor is within the snap aperture, the mouse will snap to the snap point inside that aperture. If the cursor is outside the aperture, the cursor will move to the snap aperture boundary.

The aperture (see “Aperture Size, below”) is a user-set area that determines the size of the marker for a snap location.

**Aperture Size**  **Aperture Size** is the area of influence of a snap location. When you move the mouse cursor within the aperture, the cursor will snap to that location. The larger the Aperture Size, the further away the mouse can be when snapped to a snap point.

**Instance MBB**  Snaps instances to their MBB—the minimum bounding box of all objects in the cell of the instance. See also “Global Alignment and Distribution Options” on page 336.

**Abut MBB-MBB of objects on the Icon layer**  Snaps instances to the instance’s abutment bounding box, which is the MBB of all objects in the cell of the instance on the Icon/Outline special layer. The abutment bounding box is useful when snapping to overlap markers in cells. See also “Minimum and Abutment Bounding Boxes” on page 337.

**Offset**  Offsets the snap location by the specified amount. This can be used to snap by an overlap amount. For example, if the offset set is set to 0.15 0.15, L-Edit will snap to a snap location (i.e. vertex) plus an extra 0.15 in the x direction and extra 0.15 in the y direction.

**Display Markers**  Displays the context sensitive snap markers while the user is drawing, selecting, or editing.
Object Snap Toolbar

Most of the object snapping commands can also be enabled or disabled using the **Object Snap toolbar**.

<table>
<thead>
<tr>
<th>Toolbar button</th>
<th>Marker Symbol</th>
<th>Description and Default shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Vertex snapping</strong> Shift+V.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Midpoint snapping</strong> Shift+D.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Edge snapping</strong> Shift+E.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Intersection snapping</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Center snapping</strong> Shift+C.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Quadrant snapping</strong> - Snaps the mouse to nearest quadrant on a curved edge of circles, pie wedges, tori, and curved polygons.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Pin snapping</strong> Shift+T.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>instance snapping</strong> Shift+S.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Toggles between <em>Instance MBB</em> and <em>Abut MBB</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Enable Object Snapping</strong> Alt+S or F9.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Setup Object Snapping</strong> - opens the <em>Object Snap</em> tab of <em>Setup &gt; Design</em> so you can change object snapping options. (See “Object Snap Setup” on page 307.)</td>
</tr>
</tbody>
</table>
Object Snap Shortcut Keys

Both Alt +S and F9 are shortcut keys that toggle all enabled object snapping on or off. (F9 will work even during a draw, edit or move operation, whereas any Alt hotkey will not.)

If snapping is on, for example, you can temporarily turn it off while in the middle of drawing one polygon to place a vertex within the aperture of another polygon. The toolbar is always updated to show the current snap state.

Order of Preference in Snapping

When multiple snap points exist within the aperture, the order of precedence for where snapping will occur is as follows:

**Box**

[1] Vertex  
[3] Center  
[4] Edge

**Polygon**

[1] Vertex  
[3] Edge

**Wire**

[1] Vertex of the wire’s centerline  
[2] Midpoint of the wire’s centerline  
[3] Edge of the wire’s centerline

**Circle**

[1] Center  
[2] Quadrant

**Pie Wedge, Torus, or Curved Polygon**

[1] Vertex  
[3] Quadrant  
[4] Center  
[5] Edge

**Port**

[1] Pin - Center of the port box  
[2] Vertex - Vertex of a port box  

**Instance**

[1] Endpoint  
[3] Instance boundary or origin
Selecting Objects

Selecting an object specifies that subsequent editing operations affect that object specifically. More than one object may be selected at a time.

By default, selected objects are outlined. To change the manner in which selection is displayed on any given layer see “Layer Setup” on page 89.

L-Edit provides several ways to select an object using the selection tool on the Drawing toolbar:

<table>
<thead>
<tr>
<th>Action</th>
<th>Mouse button</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicitly select an object or set of objects before an operation is performed.</td>
<td>SELECT</td>
</tr>
<tr>
<td>Implicitly select an object in the process of performing an operation on it.</td>
<td>MOVE/EDIT</td>
</tr>
<tr>
<td>Add an object to a set of selected objects.</td>
<td>EXTEND SELECT (Shift+SELECT)</td>
</tr>
</tbody>
</table>

Explicit Selection

To explicitly select an object, position the pointer over the object to be selected and click the SELECT button. Any previously selected objects are automatically deselected.

Using a Selection Box

You can also explicitly select a set of objects by dragging a selection box around them, as follows:

- Position the pointer outside the set of objects to be selected.
- Drag the pointer with the SELECT mouse button held, forming a selection marquee around the objects.
- Position the opposite corner of the selection marquee so that the marquee completely encloses all the objects to be selected but does not completely enclose any other objects, and release the SELECT mouse button.

Disallowing Instance Selection

You can set instances so they cannot be selected by right-clicking on the Instance icon of the Drawing toolbar. When

- The instance icon showing that icons can be selected.
- A red dot on the instance icon indicates that icons cannot be selected.
Chapter 8: Drawing Objects

Selecting Objects

Implicit Selection

If no other objects are selected, pressing and holding the MOVE/EDIT mouse button in or near an object within the selection range will select that object and begin a move or edit operation.

Note that implicit selection is governed by the values set for the selection range and deselection range (see “Selection and Deselection Ranges” on page 86.) Depending on these values, you may accidentally include previously selected objects (outside the deselection range) when you select another object implicitly.

To avoid this potential problem you can use Edit > Deselect All to deselect all objects before you perform implicit selection. You can also set your deselection range to be so large all objects in your design will be included.

Extend Selection

You can extend a selection by including another object or group of objects in the set of already selected objects. Select the additional object(s) with the EXTEND SELECT (Shift+SELECT) mouse button. Previously selected objects are not deselected.

Cycle Selection

When you click repeatedly with the pointer in the same spot, L-Edit uses “cycle selection” to successively select each object within your defined selection range.

The order of selection is determined by the distance from the point clicked to the nearest boundary of each object. The first click selects the nearest object, the next click moves selection to the next closest object and so on within the selection range.

Selection proceeds regardless of object type. Once each object within the selection range has been selected, the next click deselects all objects and the following click restarts the cycle beginning with the closest object.

If you click inside a single object, that object is selected first. If you click inside overlapped objects, the object with the nearest interior boundary is selected first and selection proceeds through the next nearest of the overlapped objects and then outside them, from nearest to furthest object.

Edge Selection

In addition to selecting whole objects, you can also select an individual edge of one or more objects.

To select one edge of an object, use Ctrl+click or Ctrl+right-click (SELECT EDGES mouse button).

Use Shift+Ctrl+right-click (EXTEND EDGE SELECT mouse button) to select additional edges without deselecting edges that are already selected.
To deselect a single or multiple edges, use Ctrl+Alt (DESELECT EDGE mouse button) with the right mouse button either click or use a drag-box.

You can also use Ctrl+click or Ctrl+right-click with a selection box to select one or more edges at once.

**Universal Selection**

You can select all objects in the active cell by choosing Edit > Select All or pressing Ctrl+A.

**Deselecting Objects**

*Deselecting* objects causes them to no longer be available for editing operations.

A deselecting range ensures that selected objects will not accidentally be deselected before an operation is performed (see “Selection and Deselection Ranges” on page 86.)

**Explicit Deselection**

To deselect a selected object without affecting other selected objects, place the pointer within the selection range of the object and use the DESELECT (Alt+right SELECT) mouse button.
Clicking the DESELECT button near an object which is not selected or outside the selection range of all selected objects has no effect.

**Implicit Deselection**

Clicking the SELECT button outside the selection range of selected objects automatically deselected the objects.

**Hidden Deselection**

When a layer is hidden, all selected objects on that layer are automatically deselected. This prevents hidden objects from being moved or edited. These objects remain deselected even after they are made visible again.

**Universal Deselection**

You can deselect all objects in the active cell by choosing Edit > Deselect All or pressing Alt+A.

**Deleting Objects**

You can remove objects from the layout in two ways:

- Choosing Edit > Cut, pressing Ctrl+X, or clicking the cut button ( ). The Cut command puts the deleted objects into the internal clipboard. From there they can be restored to the current cell or pasted into another cell in the same file (see “Pasting Objects” on page 328).
- Choosing Edit > Clear, or pressing Delete or Backspace. The Clear operation does not put the deleted objects into the internal clipboard. They can be restored to the active cell only with the Undo command (see “Undoing or Redoing an Operation” on page 314).

**Undoing or Redoing an Operation**

L-Edit maintains a list of edited objects and operations on a per cell basis in the undo buffer. Choosing Edit > Undo, pressing Ctrl+Z, or clicking the Undo button ( ) reverses the last operation performed in a cell. You may continue undoing your operations in reverse order, one at a time, up to and including the first operation on the cell since opening or saving it. L-Edit maintains a separate undo buffer for each cell. Only those operations that directly affect objects—drawing, copying, editing, moving, instancing, grouping, flipping, rotating, slicing, and merging—can be undone.

**Undo**

 Undo reverses mouse-based draw, move, edit, and copy operations, and the following commands:

- Edit > Cut
- Edit > Paste
- Draw > Group / Ungroup
- Draw > Rotate
- Draw > Flip > Horizontal / Vertical
- Draw > Slice > Horizontal / Vertical
- **Draw > Merge**
- **Cell > Instance**
- **Cell > Flatten**

The following operations clear the undo buffer. Editing performed prior to any of these operations cannot be reversed with the **Undo** command.

- **File > Save**
- **File > Save As**
- **Cell > Revert Cell**
- **Tools > Generate Layers**

**Redo**

You can reverse an **Undo** command by choosing **Edit > Redo**, pressing **Ctrl+Y**, or clicking the redo button ( ).

After an **Undo** operation is performed, the object or operation goes into a *redo buffer*, also maintained by L-Edit on a per-cell basis. After executing an **Undo** command, you can use the **Redo** command to revert the cell to its state before the **Undo** command was executed.

Like the undo buffer, the redo buffer is maintained separately for each cell. The redo buffer is subject to the same guidelines and restrictions as the undo buffer, and it is cleared by the same methods. When editing continues, the redo buffer is cleared. The depth of both buffers is limited only by computer resources.

**Grouping and Ungrouping Objects**

The **Group** command creates a new cell containing all objects and instances that are currently selected. This new cell is instanced into the active cell. If the selected objects are all instances of the same cell and meet additional criteria, the command will instead create an array as described in “Grouping Instances to Create an Array” on page 316.
Choose **Draw > Group** or press **Ctrl+G** to execute this command.

**Group Cell**

- Enter or select the **Name**, **View** and **Library** of the new cell.

**Origin**

- Select an origin from the pull-down menu. Note that this is a persistent setting.
  - Top Left corner
  - Top right corner
  - Bottom left corner
  - Bottom right corner
  - Center
  - Parent cell origin

**Cell info**

- Enter **Author**, **Organization**, and **Information** (notes or messages) for the new cell.

### Grouping Instances to Create an Array

Any type of object (geometry, ports, instances) may be grouped. The command can also be used to create an array from selected instances of the same cell, under certain conditions. The selected instances must be:

- Of the same cell.
- Have no repeat values.
- Have the same *orthogonal* transformations and regular translations. (Non-orthogonally rotated instances cannot be grouped.)
In other words, **Draw > Group** (Ctrl+G) can transform a collection of instances that already have the appearance and spacing of an array into a single object that L-Edit recognizes as an array. If these conditions are met, an array is automatically formed. If not, L-Edit prompts for the name of the new cell to be created from the selected objects.

**Ungrouping Instances**

**Draw > Ungroup** (Ctrl+U) flattens the selected instances into their component objects, without deleting the cell created by **Draw > Group**. When used on an array, the command “explodes” the array into its component instances.

**Draw > Ungroup** works independently of the **Group** command, and can be used to remove an array from any existing instance, however it was created. In other words, it works like **Cell > Flatten**, except that **Flatten** will flatten an entire cell, including all its instances, but **Ungroup** will flatten just one level down (only the selected instance or set of instances).

When you ungroup an array in L-Edit, each individual instance does not automatically inherit the instance name. However, if more than one instances have the same instance name before netlist extraction, L-Edit automatically assigns them unique instance names after netlist extraction by appending “_n” to each of the remaining instances, incrementing n by one for each instance.

For example, if three instances have the instance name “CAP” in the layout, after netlist extraction one of the instances will keep the instance name “CAP,” the second instance name is “CAP_2” and the third instance name is “CAP_3.”

**Undoing Draw > Group and Draw > Ungroup**

Both **Draw > Group** and **Draw > Ungroup** can be reversed with the **Undo** command.

- Executing **Undo** immediately after **Draw > Ungroup** results in the selected objects being grouped again, as if the **Draw > Group** command had just been used for the first time.
- Executing **Undo** immediately after **Draw > Group**, however, is not a complete reversal of **Draw > Group**. The cell created by **Draw > Group** is not deleted.
9 Editing Objects

You can edit objects graphically, using your keyboard and mouse. You can resize and reshape objects, perform stretch editing, add vertices to polygons or wires, and slice, merge, or nibble objects.

Finding Objects

To search for geometric objects or for ports or instances of a particular name, choose Edit > Find, press Ctrl+F, or click the find button ( ).

Find what

If no layer is specified, L-Edit searches for the checked objects on all layers.

An Instance search finds instances either by the instance name or the originating cell name as specified in the instance field.

Select a layer from the On layer pull-down menu to limit the search to just that layer.
During an L-Edit session, search parameters typed in the **Find Object(s)** dialog remain in memory and are used for all subsequent **Find** operations. The search parameters are not cleared when you switch between cells and files.

### Find Next/Find Previous

When an object has been found, you can search for the *next* object or for the *previously* found object. Choosing **Edit > Find Next**, pressing F, or clicking the find next button ( ) prompts L-Edit to search for and select the next object satisfying the current search criteria.

Choosing **Edit > Find Previous**, pressing P, or clicking the find previous button ( ) prompts L-Edit to search for and select the previous object satisfying the current search criteria.

If the **Find** command has not yet been executed, the **Find Object(s)** dialog is opened. The **Find Next** and **Find Previous** operations use the current search criteria, even if those criteria were originally set in a different cell or file.

The **Find Next** and **Find Previous** operations select objects in a cyclical manner. When the last object matching the search parameters is found, L-Edit repeats the search, beginning with the first object found.

### Hierarchical Find

The Hierarchical Find docking view can be used to search for objects through the hierarchy of the design. Results are presented in a table. Selecting an item in the table will select the corresponding item on the layout, pushing into the hierarchy if needed. The commands **Edit > Find in Hierarchy Next** and **Edit > Find in Hierarchy Prev** can be used to step through the list. These may be bound to user selectable shortcut keys.

Objects may be highlighted objects with markers as you select items in the table. Highlighting options include i) Highlight shape, ii) Display circle, and iii) Display cross hairs. The Automatic Viewport Change setting in Setup Application > Selection can be used to control panning and zooming as you select objects in the list. Automatic Viewport Change options include i) Leave view unchanged, ii) Pan to center of object(s), and iii) Zoom to objects. Objects on hidden layers are not listed, with the exception when a layer is specified in the dialog.
### Name

The name of the Port, Label, Instance or cell for the search.

### Match

- **Exact** - requires object name to match Name exactly for object to be listed.
- **Partial** - requires any part of the object name to match Name for object to be listed.
- **Wildcard** - treat * as a wildcard character when matching object name with Name.

### Where

- **Cell** - Search for Objects in the current cell.
- **Hierarchy** - Search for Objects in the current cell and hierarchy.
- **Library** - Search for Objects in all cells in the library of the currently active cell.
- **All Libraries** - Search for objects in all libraries.

---

## Moving Objects
You can reposition and reorient objects in L-Edit graphically with the mouse and keyboard; textually by entering coordinates and other values; or by using the interface or UPI macros.

**Note:** If an object is drawn on a locked layer, it cannot be edited or moved. To edit or move such an object, you must first unlock the currently locked layer or layers.

### Repositioning

To move an object, select it and position the pointer anywhere except on a vertex or edge of the selected object. Holding the MOVE/EDIT button, drag the object to its new position.

Note that the MOVE/EDIT button function depends on the position of the pointer:

- If the pointer is within the edit range set for the current design, an EDIT is performed.
- If the pointer is beyond the edit range, a MOVE is performed.

The edit range is specified using **Setup > Design—Selection**. However, you can “force” a move rather than an edit operation using the **Draw > Force Move** command (default hotkey **Alt+M**) (see “**Force Move Mode**” on page 324).

A single object can be implicitly selected and moved by clicking the MOVE/EDIT button in or near it (but not on a vertex or edge) and dragging the object to its new position. The object is automatically deselected after the move.

Multiple objects to be moved simultaneously must all be explicitly selected. The pointer may be initially positioned anywhere, including on any vertex or edge. When moved, the selected objects’ relative positions are maintained.

To constrain movement to the horizontal or vertical directions only, hold the **Shift** key down while using the MOVE/EDIT button.
Move By Options

You can move selected objects a specified distance and under various constraints using the Move dialog, which you open by choosing Draw > Move By.

Selection relative to current position
Moves the origin of the selected object(s) by the x, y offset values entered relative to the origin of their current position.

Selection by vector from base point to absolute coordinates
Moves the selected object(s) as determined by a vector from the base point position to the absolute coordinate specified by the x, y value entered. This option is only available when base point is enabled.
**Move single selected edge perpendicularly, preserving adjacent angles**

Moves the selected edge in a perpendicular direction by the distance entered in the **Distance to move edge perpendicularly field**, preserving the adjacent angles. A negative value moves the edge in the opposite direction a positive value does.

**Move single selected edge perpendicularly, preserving edge length**

Moves the selected edge in a perpendicular direction by the distance entered in the **Distance to move edge perpendicularly field**, preserving the length of the selected edge. A negative value moves the edge in the opposite direction a positive value does.

**Round offsets to mfg grid**

When checked, snaps the user-supplied x, y or distance values to the manufacturing grid when moving object(s).
Nudge

Use this feature to incrementally move (nudge) a selected object or objects a predetermined distance. The movement increment is the same in every direction. You specify nudge distance with Setup Design–Drawing.

<table>
<thead>
<tr>
<th>Command</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw &gt; Nudge &gt; Left</td>
<td>Ctrl + (left arrow key)</td>
</tr>
<tr>
<td>Draw &gt; Nudge &gt; Right</td>
<td>Ctrl + (right arrow key)</td>
</tr>
<tr>
<td>Draw &gt; Nudge &gt; Up</td>
<td>Ctrl + (up arrow key)</td>
</tr>
<tr>
<td>Draw &gt; Nudge &gt; Down</td>
<td>Ctrl + (down arrow key)</td>
</tr>
</tbody>
</table>

Force Move Mode

In normal drawing mode, the EDIT command is active when the cursor is within an object’s edit range parameter and the MOVE command is active when the cursor is outside that range.

When you use the Draw > Force Move mode (default hotkey Alt+M) L-Edit will perform a MOVE operation regardless of the cursor position (the status bar will show Mode: Move.)

If no objects are selected prior to using this command, the move adheres to the selection range for implicit selection—nothing will happen unless you are within the selection range. After the move operation is finished, L-Edit reverts to normal drawing mode (Mode: Drawing in the status bar), or you can cancel the force move operation by pressing ESC.

Base Point Mode

The base point feature enables a user-specified reference point for editing operations, which change as follows when L-Edit is in base point mode. After an editing operation is performed in base point mode, L-Edit will return to the previous mode.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move</td>
<td>Moves the selection from the base point to the current cursor location.</td>
</tr>
<tr>
<td>Edit</td>
<td>Edits (stretches) the selection from the base point to the current cursor location.</td>
</tr>
<tr>
<td>Rotate</td>
<td>Rotates the selection around the base point.</td>
</tr>
<tr>
<td>Flip</td>
<td>Flips the selection with respect to the base point.</td>
</tr>
<tr>
<td>Instance</td>
<td>Instances the cell with its origin at the base point.</td>
</tr>
<tr>
<td>Cut, Copy</td>
<td>Pastes the object at the base point. You can also use the base point to set the origin of the object being cut or copied.</td>
</tr>
</tbody>
</table>
Setting the Base Point

The base point toolbar displays the location of the base point in an editable field you can use to type in coordinates.

To simultaneously place the base point at the cursor’s current location and turn on base point mode, use Draw > Base Point > Place Base Point at cursor or the keyboard shortcut Ctrl+Q. To use your cursor to pick a base point, use the icon or Draw > Base Point > Pick Base Point.

Once a base point is set you can toggle the mode on and off with the icon , the keyboard shortcut Shift+Q, by using Draw > Base Point > Use Base Point.

Move and Copy/Paste Operations in Base Point Mode

When you cut or copy and then paste an object(s) in base point mode, L-Edit uses the base point as the reference point. When you paste in base point mode but no reference point was set explicitly during the last copy operation, the lower left corner of the copied object(s) is pasted at the base point.

The base point feature can also be used to control positioning of a pasted object during a cut or copy command. To do so, select an object, then place the base point at the desired location. The origin of the object will be pasted at the base point rather than at the middle of the layout window (the normal default).

You can also use the base point to pick an origin for an object before the cut/copy command. You must use the base point to pick a paste origin point, otherwise L-Edit will simply paste the object using its default origin in the default location.

<table>
<thead>
<tr>
<th>Copy/Cut...</th>
<th>Paste...</th>
</tr>
</thead>
<tbody>
<tr>
<td>base point ON</td>
<td>base point ON: The base point is the reference during the copy operation and that reference point is the base point during the paste operation.</td>
</tr>
<tr>
<td>base point ON</td>
<td>base point OFF: The base point is the reference during the copy operation but the center of the copied objects is pasted at the center of the screen.</td>
</tr>
<tr>
<td>base point OFF</td>
<td>base point ON: The the lower left corner of the copied objects is pasted at the base point.</td>
</tr>
<tr>
<td>base point OFF</td>
<td>base point OFF: Normal cut/paste behavior.</td>
</tr>
</tbody>
</table>

The move command (middle mouse button) moves the selected objects according to the values set in the Move dialog (see “Move By Options” on page 322).
Reorienting

These commands change the orientation of selected objects.

<table>
<thead>
<tr>
<th>Command</th>
<th>Shortcut</th>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw &gt; Rotate &gt; 90 degrees</td>
<td>R</td>
<td>![Rotate selected objects]</td>
<td>Rotates the selected object 90° counterclockwise about its geometrical center.</td>
</tr>
<tr>
<td>Draw &gt; Rotate &gt; Rotate</td>
<td>Ctrl+R</td>
<td>![Rotate dialog]</td>
<td>Opens a dialog to rotate the selected object counter-clockwise by ( n ) degrees with respect to a specified point.</td>
</tr>
<tr>
<td>Draw &gt; Flip &gt; Horizontal</td>
<td>H</td>
<td>![Horizontal flip]</td>
<td>Flips the selected object about the vertical axis through its geometrical center.</td>
</tr>
<tr>
<td>Draw &gt; Flip &gt; Vertical</td>
<td>V</td>
<td>![Vertical flip]</td>
<td>Flips the selected object about the horizontal axis through its geometrical center.</td>
</tr>
</tbody>
</table>

When multiple objects are selected, the rotation or flip occurs about the geometrical center of the selected group.

**Specifying Rotation Parameters**

The **Rotate Selected Objects** dialog allows you to specify the angle and reference point around which to rotate the selected object.
To access this dialog, select an object or objects, then choose **Draw > Rotate > Rotate** or press **Ctrl+R**.

Enter a **Rotation angle** value in degrees to specify the angular distance around which to rotate the selected object. The angle must be between -360 and +360 degrees, exclusive. L-Edit allows for six decimal-point precision in specifying the angle of rotation.

**Note:**
You can use the up and down arrows to step through multiples of 90 degrees between -270 and 270. Other values can be typed directly in the editing field.

You can also use the base point feature (see “Base Point Mode” on page 324) to specify the reference point around which L-Edit will rotate the selected object(s).

**Moving Objects from One Layer to Another**

Use **Draw > Convert > Change Layers** to transfer all objects on one layer in a given set of cells to another layer. As with any multi-cell operation, adequate cell reservations should be obtained. Note that this operation respects the cell lock status. Note also that this operation will not modify the layers in standard vias.

**Change layers of objects found in**

**From layer**
Select the scope of the operation from Selection, Cell, Hierarchy, Library or All Libraries.

**To layer**
Select the layer from which objects will be moved. (Only those layers that have geometry on them are included in the drop-down list.)

Select a layer from the list to which objects will be moved.
Copy objects to the internal clipboard by choosing Edit > Copy, pressing Ctrl+C, or clicking the copy button ( ). The copy operation saves a copy of the selected object(s) to the internal clipboard. The copied objects must be placed using the paste operation command (see “Pasting Objects” on page 328).

Duplicating Objects

Chose Edit > Duplicate, press Ctrl+D, or click the duplicate button ( ). The Duplicate operation copies the selected object(s) and pastes them, as the current selection, exactly over the original(s) in the active cell. The Duplicate command also saves the x- and y-translation when you drag the duplicated objects to a new position immediately after they are duplicated. (The duplicate objects remain selected until you explicitly select something else.)

Since the Duplicate command stores the offset as well as the objects, subsequent use of the command allows you to create regular structures quickly and accurately.

Keep in mind that multiple placement of the same object can be useful in making arrays, but it can also result in designs that use a great deal of memory and are difficult to update. Multiple placement of the same object should not be used as a substitute for good hierarchical design using instantiation.

Duplicate does not update the contents of the internal L-Edit clipboard, so objects placed on the clipboard using Copy can still be pasted to layout.

Copying Images

Large areas of the layout can be copied as a bitmap to the external Windows clipboard by choosing File > Image > Copy To Clipboard. These bitmap images can be pasted into other applications, but they cannot be pasted back into L-Edit. The resolution of the bitmap is the same as that of the screen.

Pasting Objects

L-Edit maintains an internal clipboard that stores cut and copied objects. It can be used to transfer objects between cells or between layers within a file.

Choosing Edit > Paste, pressing Ctrl+V, or clicking the paste button ( ) places the stored object(s) in the center of the active layout window, unless the Paste to cursor feature (see below) is enabled.

Choosing Edit > Paste to Layer or pressing Alt+V also places the stored object in the center of the layout window of the active cell (unless the Paste to cursor feature is enabled). In addition, this command places the object on the currently selected layer. If you select multiple objects on separate layers, they will all be pasted to the single layer specified with the Paste to Layer command. Pasted objects are automatically selected after execution of the paste command.

The contents of the internal clipboard can be pasted multiple times. Objects remain in the clipboard until another object is cut or copied, or until the file is closed.
Note: When you paste an object to a layer, L-Edit will overwrite the object’s GDSII data type with the data type of the target layer. If the target layer has no GDSII data type assigned, the pasted object will retain its original data type.

Paste to Cursor Feature

If the Paste to cursor box in the Setup Application—General dialog is checked, the contents of the clipboard appear in the layout window and move with the pointer until any mouse button is clicked. The cursor attaches to the origin of the geometry being pasted, or, if the origin is outside the instance MBB, to the point on the MBB closest to the origin.

When you execute the Paste command, the objects are positioned at the location of the cursor. Before clicking the mouse button, you can flip or rotate the objects horizontally or vertically by using the keyboard shortcut commands. (See “Reorienting” on page 326 for a list of default shortcut commands.)

You can also set a base point to control the origin of the copy or copied object and the location of the origin to which it is pasted. See “Base Point Mode” on page 324.

Resizing and Reshaping

You can change the dimensions of a box, port, or polygon by selecting and moving a vertex or an edge.

You can resize a circle by dragging its perimeter towards or away from the center to change the radius.

To resize or reshape an object, select a vertex or edge, click the MOVE/EDIT mouse button and drag the vertex or edge to the desired position.

You can resize a wire by selecting a wire edge and dragging it; and you can add vertices to a wire by selecting it and choosing Draw > Add Wire Section. Note, however, that the “perpendicular” constraint to moving edges (see “Two Options for a Perpendicular Edge Move,” below) does not apply to wires.

To change the width of a wire, you must use the Edit Object—Wire dialog (see “Editing Objects From a Dialog Window” on page 351).

Types of Edge Moves

There are three possible types of constraint when you move an edge.
Standard Move- Preserve Edge Length, Change Angle

The default move operation for an edge preserves the length of that edge but allows the angle of the adjacent edges to change, as shown below.

**MOVE/EDIT**

The red layer shows the original shape and the red vector is perpendicular to the original position of the edge.

The green layer shows a standard edge move where the edge can be moved in any direction – the length of the selected edge does not change, but the angle of the adjacent edges can.

Orthogonal Edge Move

If you use **Shift** + the MOVE/EDIT mouse button, the edge is constrained so that the anchor point can only move orthogonally with respect to the x- and y-axes.

**Shift + MOVE/EDIT**

The red layer shows the original shape and the red vector is perpendicular to the original position of the edge.

The green layer shows an orthogonal edge move where the edge can only be moved in the x- or y-axis directions – the length of the selected edge and the angle of the adjacent edges both change.

The black vector shows that in this case the move was in the positive x-direction.

Two Options for a Perpendicular Edge Move

The **Perpendicular edge move** control in **Setup Application > Editing** has two radio buttons, **Preserve angles** and **Preserve edge length**, that determine which behavior is in effect when you use **Shift + Ctrl** + the MOVE/EDIT mouse button.
Perpendicular Edge Move - Preserve Length, Change Angle

If you want to move an all-angle edge in the perpendicular direction while preserving its length, you will need to set the **Perpendicular edge move** control in **Setup Application > Editing** to **Preserve edge length**. This allows you to shrink or extend an all-angle object at one side only when you use the **Shift + Ctrl + MOVE/EDIT** mouse button.

Perpendicular Edge Move - Change Length, Preserve Angle

If you want to move an edge so that the direction of movement is perpendicular to the angle of the edge, set the **Perpendicular edge move** control in **Setup Application > Editing** set to **Preserve angles**.

When you use **Shift + Ctrl +** the MOVE/EDIT mouse button the edge will retain its original angle.

For multiple adjoining edges, the perpendicular direction is taken to be relative to the line connecting the first and last vertex in the collection of edges.

Note that this move operation snaps vertices to the manufacturing grid. Note that due to this “perpendicular snapping,” two separate moves may not produce exactly the same result as a single move.

Editing Multiple Objects Simultaneously

You can resize or reshape one or more boxes, polygons, wires, pie wedges, tori, or ports simultaneously by selecting and moving sets of their edges.

Select the edges of the desired objects. To modify the selected objects, drag the edges in the desired direction with the MOVE/EDIT mouse button. All selected edges and objects will move the same direction and distance, subject to any constraints imposed by the objects themselves.
Adding Vertices

You can add vertices to all-angle polygons or wires after you create them. To add a vertex, select the object and position the pointer on the edge where you want the new vertex to be. Hold Ctrl+the MOVE/EDIT mouse button while dragging the new vertex into position.

Pie Wedges and Tori

When you use the mouse to reshape and resize a pie wedge or torus, you can change the sweep angle or the radii. To change the sweep angle, position the pointer on one straight edge of the selected object, click the MOVE/EDIT mouse button and drag the mouse in the desired direction. To change the radii, place the pointer on the curved edge and drag it to the desired position. You can change either radius of a torus.

Resizing Text

Use Draw > Convert > Resize Text to either scale port or label text size or set it to a specific size.

![Resize Text Dialog Box](image)

**Resize Text found in**
Select the scope of action for this cell change, and enter a check to enable changes to **Labels** and/or **Ports**.

**Scale Text**
Scales text size. The default value is 50%. Scale values less than 100 decrease the text size; values above 100 increase the text size.

**Fixed Size**
Enter a text size in locator units. The default value is the default text size for the active TDB file.

**Statistics**
Displays the minimum, average, and maximum port text sizes used in the active file.

To add text as a drawn element of a cell, please see “Adding a Copyright, Logo or Text to a File” on page 126.
Measuring Distance Between Objects

Temporary Rulers

L-Edit provides a temporary ruler in the form of a marker that can be used with any drawing tool. The temporary ruler can measure in any angle and direction allowed by the current drawing tool, starting at the location of the cursor when it is activated.

Use the shortcut key T key to initiate or end drawing a temporary ruler. The left mouse button ends measurement and retains the display of the ruler you are actively drawing. The right mouse button ends measurement and erases the ruler that you are actively drawing.

Temporary rulers respond to the key control Shift to force 90° drawing, Ctrl to force 45° drawing and Ctrl+Shift to force all-angle drawing.

You can use the Toggle Markers button to toggle display of temporary rulers and the Clear Markers button to erase temporary rulers – along with all other markers. These buttons are on both the Verification Navigator and the Node Highlighting toolbar.

There is a Temporary rulers text height field in Setup > Application—Rendering where you can set the text size for temporary rulers.

Tools > Measure

The measure command displays the minimum distance between any two wires, boxes, polygons or instances. It allows you to measure objects at the same or across different levels of cell hierarchy.

Measurements are displayed to three decimal points, as a marker that is a double-ended arrow, so you can toggle their display on and off, or delete them along with all other markers in the cell.

L-Edit shows the shortest distance when i) an edge (or multiple edges) on two different objects are selected, ii) an edge (or multiple edges) on one object and a different object are selected, or iii) two objects are selected.

When more than one edge or an entire object is selected, L-Edit measures the shortest distance between objects. (36.385)

When a single edge of an object is selected, the measurement is applied from that edge. (91.791)
Measuring Between Objects at the Same Level of Hierarchy

[1] Select the first object or one of its edges, then click on **Tools > Measure > Set Measurement Reference**.

[2] Select the second object or one of its edges and click on **Tools > Measure > Between Selections**.

L-Edit will display the measurement of the shortest distance between your selections.

Measuring Between Objects on Different Levels of Hierarchy

When you select objects at different levels in the hierarchy, the measurement marker is only displayed on the hierarchy level of the last selected object. The cell in which you set “To Current Selection” is the level at which the measurement is displayed.

[1] In the higher-level cell, select the first object, object edge, or instance. If your first selection is an instance, use **Page Down** to push down to its original cell and select an object or object edge in the original cell.

[2] Click on **Tools > Measure > Set Measurement Reference**.

[3] In the higher-level cell, select the second object, object edge or instance, using **Page Up** if necessary. If your second selection is an instance, use **Page Down** to push down to the original cell and select an object or object edge in the original cell. Click on **Tools > Measure > To Current Selection**.

[4] Use **Page Up** if necessary to view the higher-level cell.

If you set the measurement reference in the lower cell, L-Edit will display the measurement in the higher-level cell. Use **Page Up** to push up the hierarchy to see the measurement.

If your first selection is in the lower cell, the measurement is displayed only in the higher-level cell.

In this case, there are two instances of Cell4 in the higher-level cell, U28 and U26.

You must push up the hierarchy to see the measurement referenced from instance U26.
Similarly, if you set the measurement reference in the higher cell, L-Edit will display the measurement in the lower-level cell. You must use **Page Down**, or open the instance original, to see it.

If your first selection is in the higher cell, L-Edit displays the measurement only in the lower-level cell. In this case, we are pushed into instance U28 of Cell4, so we can see the measurement referenced from the higher-level cell.

When we push into instance U26 of Cell4, we see the measurement referenced from the higher-level cell to instance U28. (In this case, the angle of the measurement is different because instance U26 is flipped and rotated.)

---

**Aligning and Distributing Objects**

You can align, distribute, or tile objects using the alignment commands from either the menu using **Draw > Align** as shown below, or the Alignment toolbar.
Alignment of objects other than instances is performed with respect to the last of all the objects selected. You can use the shortcut Ctrl + Space to cause L-Edit to highlight the last object selected (or the command Draw > Align > Flash Last Selected Object.)

<table>
<thead>
<tr>
<th>Left: Mode</th>
<th>Right: Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Center &amp; Middle</td>
<td>Top</td>
</tr>
</tbody>
</table>

**Global Alignment and Distribution Options**

Use Draw > Align > Options to set global parameters for the relative alignment of objects and the grid to which they are snapped.

![Alignment Options Dialog]

**Bounding Box for instances**

Controls how instances are selected for aligning, tiling, or distributing (see “Minimum and Abutment Bounding Boxes,” below).

- **Minimum bounding box** — use the minimum bounding box of all objects in the cell.
- **Abutment bounding box** — use the minimum bounding box of all objects in the cell on the Icon special layer.

**Alignment done relative to**

Controls which set of objects are included in the MBB that acts as the point of reference for alignment operations.

- **MBB of selected objects** — use the MBB of the currently selected objects.
- **Last selected object** — use the MBB of the last selected object.

**Snap alignment point & distribution distance to**

When checked, prevents objects from being placed off-grid. All alignment points and distribution distances will be snapped to either the Manufacturing grid or the Mouse snap grid.
Minimum and Abutment Bounding Boxes

For alignment, tiling, and distributing, you have the option to use either the minimum bounding box (MBB) of all objects in the cell or the abutment bounding box which is the MBB of all objects in the cell on the Icon/Outline special layer. The abutment bounding box is useful when tiling standard cells that need to be overlapped.

Alignment Commands

- **Align Left** - Aligns the left edge of objects to the left edge of the bounding box (MBB) of the selected objects or the left edge of the last selected object.

- **Align Horizontal Centers** - Aligns objects vertically to the center of the MBB of the selected objects or the center of the last selected object.
**Align Right** - Aligns objects to the right edge of the MBB of the selected objects or the right edge of the last selected object.

![Align Right](image)

**Align Vertical and Horizontal Centers** - Aligns the horizontal and vertical centers of objects to the center of the MBB (of either the selected objects or the center of the last selected object).

![Align Vertical and Horizontal Centers](image)
Chapter 9: Editing Objects

Aligning and Distributing Objects

**Align Tops** - Aligns objects to the top edge of the MBB of the selected objects or the top edge of the last selected object.

**Align Vertical Centers** - Aligns objects horizontally to the center of the MBB of the selected objects or the center of the last selected object.

**Align Bottoms** - Aligns objects to the bottom edge of the MBB of the selected objects or the bottom edge of the last selected object.

---

**Global Distribution Options**

The anchor of a distribution operation is determined by the objects having a MBB with the least \( x \) value and then the objects having a MBB with the least \( y \) value. If two or more objects have the same minimum \( x \) or \( y \) value for their MBB, the largest of these objects is given the left most or bottom most position.
If two or more objects have the same minimum $x$ or $y$ value for their MBB and are the same size, the object whose layer is highest in the layer list is selected for the left most or bottom most position.

**Distribute Distance**

- **Distance**—Spaces the objects by a specific amount.
- **Minimum distance between objects**—Spaces the objects by the minimum distance or pitch between all objects.
- **Maximum distance between objects**—Spaces the objects by the maximum distance or pitch between all objects as the value to space the objects.
- **Average distance between objects**—Spaces the objects by the average distance or pitch between all objects as the value to space the objects.
- **Distance between the first two left/bottom objects**—Spaces the objects by the distance or pitch between the two objects that are the left most, bottom most objects.

**Distribute by**

- **Pitch**—Evenly spaces the center of the objects’ MBB
- **Edge to Edge**—Evenly spaces the outside edge to outside edge of the objects’ MBB. (See illustration that follows.)
Distribution Commands

**Distribute Horizontally** - Equally spaces all objects horizontally starting with the left most, bottom most object. When you run this command, a dialog will appear asking how you want to space the objects. No alignment is performed on the distributed objects.

**Distribute Vertically** - Equally spaces all objects vertically starting with the left most, bottom most object. When you run this command, a dialog will appear asking how you want to space the objects. No alignment is performed on the distributed objects.

Tiling Options

Objects for tiling are sorted into rows and columns according to a MBB around them. For two objects to be on the same row, the midpoint of each object has to be within the MBB of the other object. In the
example below, the midpoint of the central blue box is above the MBB of the lower green and red box, so it will be on the second row.

Tile Commands

Tile Horizontally - Tiles or stacks the objects horizontally so that each objects is next to each other with the bottom of their MBB’s aligned. For sorting order, see Distribute Horizontally.

![Tile Commands Image]
Tile Vertically - Tiles or stacks the objects vertically so that each object is next to each other with the bottom of their MBB’s aligned. For sorting order, see Distribute Horizontally.

Tile as a 2D Array - Tiles or stacks the objects horizontally so that each object is next to each other with the bottom of their MBB’s aligned.

Slicing

Divide selected objects along a horizontal line by choosing Draw > Slice > Horizontal or clicking the horizontal slice button ( ). Divide selected objects along a vertical line by choosing Draw > Slice > Vertical or clicking the vertical slice button ( ).

Note: Ports, rulers, and instances cannot be sliced. If you select objects of these types, L-Edit ignores them during a slice operation.
Divide selected objects along a line at any angle by choosing **Draw > Slice > All Angle**, then specify the slice line in the dialog.

![Slice Dialog](#)

You can specify the angle of the slice by entering the x- and y- coordinates of two points to define a slice line, using the **Pick** buttons to set the two Point 1 and Point 2 graphically, or by specifying one point and entering an **Angle** value.

When you execute a slice command, the view automatically zooms to include all selected objects and a horizontal or vertical line appears, indicating where to slice (divide) the objects. The line moves with the pointer until you place it by clicking any mouse button, at which time each object splits into two new objects with coincident edges.

**Note:** When slicing, circles, pie wedges, tori, and curved polygons are approximated by an all-angle polygon within the tolerance set in manufacturing **Setting and Displaying the Grid Layer**.

**Nibbling**

To **nibble**, or cut out, an area from selected objects in the active cell, perform the following steps:

- Select the desired objects. They may be on multiple layers.
- Select the drawing tool to use for nibbling.
- Choose **Draw > Nibble**, press **Alt+X**, or click the nibble button ( ) to draw the shape to nibble from the selected objects. An area equal to the shape is deleted from the objects.

You can only nibble certain objects. These are the same objects you can use as a nibbling tool. These objects are:

- Box
- 90°, 45°, all-angle, and curved polygon
- 90°, 45°, and all-angle wire
- Circle
- Pie Wedge
- Torus
L-Edit ignores attempts to nibble ports, rulers, and instances.

When you use a wire as a nibbling tool, the default wire width for the Drag Box Layer must be set to the width of the nibbling wire. (To change wire parameters for a specific layer, see “End Styles and Join Styles” on page 297.) If the wire width for the Drag Box Layer is set to zero, you will not be able to use wires to nibble other objects.

**Note:**
For nibble operations, circles, pie wedges, tori, and curved polygons are approximated by an all-angle polygon within the tolerance set for the manufacturing grid (see “Setup Design–Grid” on page 82.)

### Merging Objects

Use **Draw > Merge** to merge multiple selected intersecting boxes, polygons (90°, 45°, all-angle, and curved polygons), or wires (90°, 45°, and all-angle), circles, pie wedges, and tori into one object. You can only merge intersecting objects that lie on the same layer. If you select objects from more than one layer, L-Edit merges each set of overlapping selected objects on the same layer into one object.

For merge operations, circles, pie wedges, tori, and curved polygons are approximated by an all-angle polygon within the tolerance set for the manufacturing grid (see “Setup Design–Grid” on page 82.)

When a wire is merged with another object it becomes a polygon.

**Warning:**
If you merge intersecting objects with different GDSII data types, L-Edit replaces their respective data type values with the data type for the layer (or to “0” if a data type has not been set for the layer) without a warning.

### Boolean and Grow Operations

You create new polygons by applying logical operations to one or more drawn objects using **Draw > Boolean/Grow Operations**. Note that L-Edit does not support Boolean operations on instances.

To perform a Boolean operation, select one or more objects on the layout and then choose **Draw > Boolean/Grow Operations** from the L-Edit menu (or use the shortcut B). Valid objects for this operation include boxes, all-angle polygons, wires, circles, pie wedges, and tori. When you click **OK** to perform the Boolean operation, L-Edit creates one or more polygons on the **Result** layer.
The **Inputs** field shows the number and type of objects selected on each layer.

The operations **And**, **Xor**, **Subtract (A-B)**, and **Subtract (B-A)** require that exactly two inputs be specified. If this condition is not met, the **Inputs** field displays an error message; you must select exactly two inputs or choose another operation. You can specify two inputs, A and B, in the following ways:

- Select exactly two objects on the same or different layers. L-Edit assigns the input names A and B to the objects, and displays these assignments in the **Inputs** field.
- Select any number of objects on exactly two layers. L-Edit assigns all selections on one layer to input A, and all selections on the other layer to input B. The corresponding layer for each input is shown in the **Inputs** field.

In the **Result** field, select the layer on which to create the resulting polygon(s).

To replace the input objects with the resulting polygon(s), check the option labeled **Delete inputs after operation is done**. To preserve all input objects, leave this box unchecked (default).

In the **Operation** field, choose one of the following Boolean operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>And</td>
<td>A &amp; B</td>
<td>Takes the intersection of inputs A and B. This function can be applied to exactly two objects, or to any number of objects on exactly two layers.</td>
</tr>
<tr>
<td>Xor</td>
<td>A ⊕ B</td>
<td>Represents the area occupied by exactly one input (A or B), excluding all areas of intersection. This function can be applied to exactly two objects, or to any number of objects on exactly two layers.</td>
</tr>
<tr>
<td>Not</td>
<td>¬A</td>
<td>Represents the outside, or inverse, of all input objects within the region defined by their collective minimum bounding box.</td>
</tr>
</tbody>
</table>
### Operation Description Illustration

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grow</td>
<td>Takes the union of all input objects, then displaces each edge outward by the distance specified in the By field. (Default: 1.000)</td>
<td>![Grow Illustration]</td>
</tr>
<tr>
<td>Shrink</td>
<td>Takes the union of all input objects, then displaces each edge inward by the distance specified in the By field. (Default: 1.000)</td>
<td>![Shrink Illustration]</td>
</tr>
<tr>
<td>Subtract (A-B)</td>
<td>A &amp; B</td>
<td>![Subtract (A-B) Illustration]</td>
</tr>
<tr>
<td></td>
<td>Represents the portion of input A that does not intersect with input B. This function can be applied to exactly two objects, or to any number of objects on exactly two layers.</td>
<td></td>
</tr>
<tr>
<td>Subtract (B-A)</td>
<td>B &amp; A</td>
<td>![Subtract (B-A) Illustration]</td>
</tr>
<tr>
<td></td>
<td>Represents the portion of input B that does not intersect with input A. This function can be applied to exactly two objects, or to any number of objects on exactly two layers.</td>
<td></td>
</tr>
</tbody>
</table>

### Converting Curved and other Objects to Polygons

You can convert boxes, wires, circles, pie wedges, tori, and polygons with curved sides to straight sided polygons. First select the objects, then choose Draw > Convert > To Polygon.

Circles, polygons with curved edges, the curved parts of pie wedges and tori will be approximated by creating new vertices and snapping these vertices to the manufacturing grid. (Vertices are added within a distance of less than the manufacturing grid of the original shape. Vertices on non-curved segments are not modified.)

Change Layers...
To Polygon
Straighten Curved Segments
Snap to Manufacturing Grid...
Fracture Polygons...
Resize Text...
Connect Segments...
To Curved Polygon
Polygon To Orthogonal/IG
**Note:**
L-Edit automatically converts circles, pie wedges, tori, and curved-sided polygons to straight-sided polygons when writing out GDSII. When writing out CIF, L-Edit converts the same shapes except circles.

### Converting All-Angle Polygons to Curved Polygons

The **Draw > Convert > To Curved Polygon** function searches a selected all-angle polygon for arcs with eight or more vertices, and replaces those multiple adjacent segments with a single curved edge. These vertices must lie on an arc with no more than one manufacturing unit radius error.

### Converting Polygons to Orthogonal or 45° Geometry

You can convert a straight-edged polygon to a polygon with either orthogonal or 45° edges using the **Draw > Convert > Polygon To Orthogonal/45** command.

The **Orthogonal** and **45’s** commands convert polygons using a predetermined step size.
Draw > Convert > Polygon To Orthogonal/45 > 45’s

Selecting either **Orthogonal Step** or **45 Step** lets you control the step size of the new edges. The angle, however, remains fixed.

Draw > Convert > Polygon To Orthogonal/45 > Orthogonal step

Before - polygon with straight edges

After - polygon with 45° edges

Smaller orthogonal step size.

Larger orthogonal step size.
Removing Curves from Polygons

You can straighten the curved segments of polygons using Draw > Convert > Straighten Curved Segments. This operation will remove the curvature from all segments of all selected polygons. Circles, pie wedges and tori are not affected.

Fracturing Polygons

The Draw > Convert > Fracture Polygons command divides polygons with a large numbers of vertices into several polygons with fewer vertices. The polygons are fractured according to the maximum number of vertices you specify.

Note that this operation does not modify wires, circles, pie wedges, or tori.

Fracture Polygons

Fracture polygons found in:
Cell: "Bulk1G51"

Maximum number of vertices: 155

Fracture
You can choose to fracture only the polygons you have selected, all polygons in a cell, all polygons in the entire cell hierarchy or all polygons in the library that contains the cell.

Maximum Number of vertices
Sets the maximum number of allowable vertices for the polygons you have selected.
Editing Objects From a Dialog Window

You can use text and data values to edit one or more drawn objects in L-Edit using the **Edit Object(s)** dialog.

**Edit > Edit Object(s)**

Select one or more objects in layout and use one of the following commands. If multiple objects are selected they will all acquire the values you enter in this dialog.

- choose **Edit > Edit Object(s)**
- press **Ctrl+E**
- double-click the MOVE/EDIT mouse button
- click the edit object(s) button ( ).

Common Text Edit Fields

The **On layer**, **GDSII Data type** and **Properties** fields are common to all **Edit Object** dialogs.

Note that if you change the layer of an object in the **Edit Object(s)** dialog but not the GDSII datatype, L-Edit will automatically change the datatype to the value of the new layer as well.

**On layer**

The layer on which the selected object(s) reside. You can select from the drop-down list to change the layer.

**GDSII Data type**

An integer ranging from 0-63, used primarily by GDSII database users who intend to export a GDSII file from L-Edit and read it into another program requiring additional information. Use this field to assign or reassign a GDSII data type value to the selected object(s).

**Properties**

Opens the **Properties** dialog for the selected object (see “Properties” on page 123). This button is only available when a single object is selected.

**Editing Multiple Object Using Text**

When you select multiple objects you can use the **Edit Object(s)** dialog to modify those objects simultaneously.
Each tab in the **Edit Object(s)** dialog indicates the number of selected objects of that type in parentheses.

### Mixed-Value and Disabled Fields

When multiple objects with different properties are selected, the affected fields will have a *mixed-value appearance*—that is, they do not contain a value since multiple values exist for those fields, and the field background is gray.

Unlike disabled (“grayed-out”) fields, which you cannot edit, mixed-value fields accept new data. All selected objects take on the value entered. In the example above, if you entered “8” in the X Translation field, all 7 of the selected instances would take the translation value 8 Microns.

### Stepping Through Selected Objects One at a Time

You can step through selected objects one at a time, by pressing the One by one.. button on the Edit Object(s) dialog. After pressing the One by one.. button, arrows appear in the dialog to step forward and backwards through the list of selected objects. Changes made in the dialog prior to pressing One by one... are applied to all selected objects when the One by one button is pressed, and cannot
subsequently be cancelled. When either arrow button is pressed, changes in the dialog are applied to the current object in the dialog and cannot be cancelled.

Objects can be highlighted with markers as you step through the selections. Highlight options include i) Highlight shape, ii) Display circle, and iii) Display cross hairs. The Automatic Viewport Change setting in Setup Application > Selection can be used to control panning and zooming as you step through the list of objects. Automatic Viewport Change options include i) Leave view unchanged, ii) Pan to center of object(s), and iii) Zoom to objects.
Edit Object(s) for Boxes

To change the coordinates or dimensions of a box, choose Edit > Edit Object(s) (shortcut Ctrl+E).

On layer, GDSII Data type, Properties

See “Common Text Edit Fields” on page 351.

Show box coordinates, using:

Choose from six methods for displaying box coordinates and dimensions in the pull-down menu. The options are Corners, Center and dimensions, Bottom left corner and dimensions, Top left corner and dimensions, Top right corner and dimensions and Bottom right corner and dimensions.

Coordinates (display units)

Coordinates of the selected box or boxes, in display units, in format determined by above selection.

(read-only fields in the bottom half of the dialog)

These read-only fields display the coordinates and dimensions derived from the values entered in the Coordinates group, in an alternate format.
Edit Object(s) for Polygons

This dialog allows you to add, delete, or modify vertices and curves.

**Vertices** (display units) The X and Y coordinates of the vertices of the selected polygon. For convenience, the pound sign (#) assigns a number to each vertex. Note that if multiple polygons are selected, this list is disabled.

**Add Vertex** Adds a new vertex with the selected coordinates.

**Delete Vertex** Removes the selected vertex from the polygon.

**Show Curve Height** If checked, a column for Curve Height is displayed in the Vertices list (see “Curve Height” on page 286.)
Edit Object(s) for Wires

Use this dialog to modify, add, or delete a wire’s vertices, or change a wire’s width, end, or join style.

**Vertices** *(display units)* The X and Y coordinates of the selected wire’s vertices. Note that if multiple wires are selected, this list is disabled.

**Add Vertex** Adds a new vertex with the selected coordinates.

**Delete Vertex** Removes the selected vertex from the polygon.

Note that values in the following four fields will override those set in **Setup > Layers**. Refer to “End Styles and Join Styles” on page 297 for more detailed information about these options.

**Wire width** The width of selected wires, in display units.

**Join style** The type of join for the wires. A drop-down menu lists the four styles: **Layout**, **Round**, **Bevel**, or **Miter**.

**End style** The type of end for the wires. A drop-down menu lists the three styles: **Butt**, **Round**, or **Extend**.

**Angle** The angle between two segments in a miter style join.
**Edit Object(s) for Circles**

Use this dialog to modify the coordinates of the center and radius of a circle.

![Edit Object(s) dialog](image)

**Coordinates (display units)**

Individual coordinate fields include:

- **X** and **Y** coordinates for the center of the selected circle
- **Radius** of the selected circle
Edit Object(s) for Pie Wedges

You can edit the center of a pie wedge, its sweep angle, and its radius with this dialog.

**Center Coordinates**
(display units)

The X and Y coordinates of the center of the selected pie wedge.

**Sweep Angle**
(Counterclockwise)

The angle from the horizontal (0°) to the **Begin** and **End** of the pie wedge.

The **Sweep Angle** is calculated counterclockwise as the angle from the horizontal (0°) to the **Begin** and **End** of the pie wedge. In the illustration below, the **Begin** angle is 90° and the **End** angle is 180°.

**Radius**

The radius of the pie wedge.
Edit Object(s) for Tori

Use this dialog to modify the coordinates of the center of a torus, its sweep angle, and its radii.

**Center Coordinates**  
(display units)  
The X and Y coordinates of the center of the selected torus.

**Sweep Angle**  
(Counterclockwise)  
The angle from the horizontal (0°) to the `Begin` and `End` of the selected torus. See “Sweep Angle (Counterclockwise)” on page 358.

**Radii**  
`Radius 1` is the first radius you create, whether it is to be the inner or outer edge of the torus. `Radius 2` is the second radius drawn.
Edit Object(s) for Ports

To edit the attributes of a port, select a port and use **Edit Object(s)**.

**Terminal name**

The associated port name.

**Terminal type**

(This attribute is not currently supported by L-Edit, but the setting is retained and editable in the OpenAccess database to integrate with tools that use these settings.)

**Port access direction**

Check these boxes to allow port access for routing at the **Top**, **Bottom**, **Left** or **Right**. (This attribute is not currently supported by L-Edit, but the setting is retained and editable in the OpenAccess database to integrate with tools that use these settings.)

**Text position**

Click on one of these boxes to position the anchor of the port label, and select either **Horizontal** or **Vertical** for the text direction.

**Coordinates**

Use these optional fields to position the anchor point of a port shape by entering absolute coordinates.
Note: If you change layers while creating a port, the GDSII data type will change accordingly. If you change layers while editing a port, however, the GDSII data type will not change unless you explicitly reset it.

Edit Object(s) for Text Labels

- **Name**: The label text.
- **Text size**: Size of the text in layout.
- **Label location**: The X- and Y-coordinates of the text anchor.
- **Text justification**: This group sets the position of the anchor with respect to the text, which can be any combination of **Left**, **Center** or **Right** and **Above**, **Middle** or **Below**.

The pull-down menu lets you set whether the text orientation is **Vertical** or **Horizontal**.

Default text placement is horizontal, centered and in the middle of the text.
Editing

L-Edit includes a user interface that allows you to use basic textual commands and their associated coordinates. It is useful for entering precise and repeatable drawing and editing operations.

You can also use it to perform command scripting with text files (see “Command Scripting” on page 373 for details).

Opening the Command Window

To open the window, select Tools > Activate. The window will initially appear docked at the bottom of the application window, but can be moved by dragging any corner or edge.

You can also use the ` key (grave accent, found below the tilde (~) symbol) to toggle the window open and closed.

Using the Command Window

Click inside the window to make it active. Commands are entered at the command prompt (a flashing cursor). Commands apply only to the layout window that is currently active.

The command window logs previous commands. Use the up and down (↑, ↓) arrow keys to scroll through prior commands for display and reuse. Text can be copied and pasted to form new commands. Use the Esc key to cancel a command.

A right-click in the window opens a context-sensitive menu for performing the following functions:

- **Paste**: Paste from Windows clipboard into window. Only first command is pasted if multiple lines were previously copied.
- **Copy**: Copy selected text from window into the Windows clipboard. Text can then be pasted into another editor or into the Command window.
- **Copy to file**: Save selected text to a .tco (Tanner command file) script file. For more information on object properties, see “Command Scripting” on page 373.
**Delete last line**  
Delete the last line of text.

**Clear all**  
Cleans all commands from the command log.

**Customize**  
Opens the **Customize** dialog (see below).

**Customize** is a dialog for configuring font size, color and background of the window. It also allows specification of a default directory for command scripts.

![Customize Command Line](image)

**TCL Commands**

As a convenience, L-Edit includes a set of TCL commands specific to the product. For example, `circle` will draw a circle with a given center point and radius on a given layer.

To list the available TCL commands simply type `help` in the command window.

The TCL commands included are often updated; here is a sample list from version 16.00:

**TCL commands specific to L-Edit:**
- `barcode`: create barcode from text
- `box`: create a box
- `cell`: open a cell (create if necessary)
- `circle`: create a circle
- `colorpalette`: query or modify color palette entries
- `cross`: create a cross
- `database`: manage and query the database
- `delete`: delete an instance, or the current selection
- `drc`: run drc on current cell
- `dtos`: double to string
- `exit`: exit command
- `findinstance`: find matching instances
- `findoverlap`: find overlapping cells
- `generatelayer`: generate derived layer(s)
- `help`: Display this help screen
- `hershey`: create geometry from text
instance: create an instance or an array
layer: select drawing layer (create if necessary)
locate: find the location of a named point
logmessage: write messages to log window
merge: merge objects on specified layers (or selection)
mgc_rve_context: specify cell for highlighting data
mgc_rve_export_layout: export layout
mgc_rve_get_location: input point
mgc_rve_get_rectangle: drag to select box
mgc_rve_highlight_index: set index of highlight color
mgc_rve_import: import layout
mgc_rve_layer: set highlight layer
mgc_rve_rect: highlight rectangle
mgc_rve_text: text on error layer
mgc_rve_text_clear: delete text on error layer
mgc_rve_zoom: zoom to box
moveorigin: relocates the origin of a cell
mr: highlight rectangle
pastetolayer: paste selection to layer
polygon: create a polygon
port: create a port
position: (re)position an element
probe: test crossprobing
property: property command
rect: create a rect
ring: construct a ring around a (list of) instance
save: save a file
select: select an object, or a layer
stod: string to double
surround: surround a (list of) instance
test: test experimental commands
ungroup: ungroup an instance, or the current selection
window: window command
wire: create a wire

Syntax

The basic command syntax is as follows:

```
command  <arguments>  <options> <press Enter> <mouse click>
```

Coordinate Entry Options

Arguments typically include a list of coordinates. Coordinates are relative by default (x1, y1). An exclamation point is used to designate absolute coordinates (!x, !y). Display units are used for relative and absolute coordinates.

For commands that support the entry of multiple coordinate pairs, the use of absolute coordinates sets the reference point for the relative coordinates that follow. For example:

```
box !x1 !y1 x2 y2
```

will draw a box with its reference point at absolute location x1,y1 and opposing corner at (x1 + x2, y1 + y2).

A -! used in an argument specifies that all values that follow are in absolute coordinates. For example:

```
box -! x1 y1 x2 y2
```
will draw a box with one corner at absolute location \( x_1, y_1 \) and the opposing corner at absolute location \( x_2, y_2 \).

**Command Completion Using a Mouse Click**

Some commands support coordinate entry using the mouse. In this case, after the partial command is typed it must be followed by the Enter key and then one or more mouse clicks to complete the command.

When multiple mouse clicks are required you use a left mouse button click to enter multiple coordinate locations, and a right mouse button click to complete the coordinate entry and execute the command.

Once the command is executed, the coordinate values entered using the mouse are displayed in the window. The Esc key can be used during mouse entry to abort a command.

**Reference Point Location**

The commands copy, move, paste, and rotate require a reference point for locating the new object position. In each command, you can keep the default reference point, specify an offset from the default reference point, or specify the reference point in absolute coordinates. The default reference point differs depending upon the command and whether or not multiple objects are selected.

When a single primitive object is selected for copy, move, or paste commands, the default reference point is the lower left corner of the minimum bounding box.

When a single instance is selected for copy, move, or paste commands, the default reference point is the instance origin.

When a single primitive object or instance is selected for the rotate command, the default reference point is the center of the instance. If the instance vertices are on the manufacturing grid or if its width and height are on the manufacturing grid but its center is off grid, L-Edit will maintain this state after rotating it. That is, the reference point will not be snapped to the grid.

**Reference Point Location for Multiple Objects**

When multiple primitive objects are selected for copy, move, or paste commands, the reference point is determined by a mouse click. In this case, the reference point may be specified at any location within
the minimum bounding box for the objects. As with other mouse click operations, you must press the
**Enter** key to pick a position using the mouse.

![A mouse click determines the reference point that will be used for multiple selected objects.](image)

When any number of primitive objects are selected for the **rotate** command, the default reference point is the center of the minimum bounding box for the objects.

![For rotate commands, the default reference point is always located at the center of the minimum bounding box for the selected objects.](image)

**Special Characters**

The special characters slash (/), space ( ), and quotes (") cannot be used in an argument. To use these characters you must either enclose the entire string in double quotes or precede each individual special character with the backslash (\) escape character.

For example, the layer name **Not Poly** may be entered as "**Not Poly"" or Not Poly. Similarly, to include a pathname C:\TEMP\NEWFILE.TDB you would enter "C:\TEMP\NEWFILE.TDB" or "C:\TEMP\NEWFILE.TDB".

If a cell name includes a space or comma, it must be enclosed in quotes.

**Keyboard Shortcuts**

The **TAB** key functions as a position-sensitive shortcut within the command window. Depending on the cursor position in a command, **TAB** will cycle through the available values for commands, layer names, cell names, and file names. For example, in the command:

```
box !24 !14 !4 !1 -1 <TAB>
```

pressing the **TAB** key will display the defined layer names in layer list order.

Typing a letter prior to using the **TAB** key will scroll the display to the first list element starting with that letter.

**Command Reference**

The following commands are available for the .
!!

Repeat and execute the last command. If the command requires mouse entry, that portion of the command is not executed.

↑, ↓

Scroll up or down through the log of command entries.

<Esc>

The Esc key cancels the current command.

Array

Create an array of the selected instance(s) with the designated number of rows and columns. The argument Xp Yp determines the distance between the origin of each element in the array.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>array Cols Rows Xp Yp</td>
<td>array 4 6 10 14</td>
</tr>
</tbody>
</table>
For positive pitch, array is performed in the positive $x$ and $y$ direction. For negative pitch, the array is performed in the negative $x$ and $y$ direction regardless of overlap, as shown below.

![Diagram of positive and negative pitch arrays](image)

**Box**

Draw a box as follows. The option -l changes the designated layer.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>box w h &lt;press Enter&gt;</code></td>
<td>&lt;mouse click&gt; Draw a box of width $w$ and height $h$ with its center at the mouse click position.</td>
</tr>
</tbody>
</table>

Example: `box 4 10 -l Metal1 <mouse click>`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>box lx1 ly1 &lt;press Enter&gt;</code></td>
<td>&lt;mouse click&gt; Draw a box with corners at $(x_1, y_1)$ and the mouse click position.</td>
</tr>
</tbody>
</table>

Example: `box 4 10`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>box lx1 ly1 lx2 ly2</code></td>
<td>Draw a box with corners at absolute locations $(x_1, y_1)$ and $(x_2, y_2)$.</td>
</tr>
</tbody>
</table>

Example: `box 42 51 46 61`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>box lx1 ly1 lx2 ly2</code></td>
<td>Draw a box with corners at absolute location $(x_1, y_1)$ and relative location $(x_1 + x_2, y_1 + y_2)$.</td>
</tr>
</tbody>
</table>

Example: `box 42 51 49 62`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>box -lx1 y1 lx2 y2</code></td>
<td>Draw a box with corners at absolute locations $(x_1, y_1)$ and $(x_2, y_2)$.</td>
</tr>
</tbody>
</table>

Example: `box -l 42 51 49 62`
**Copy**

Copy the selected object(s) as follows. The option `-l` changes the designated layer. The option `-R` followed by two relative coordinates specifies an offset of the reference point from the lower left corner of the minimum bounding box.

If multiple objects are selected, this command prompts for a mouse click to determine the reference point that will be used when the objects are pasted.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy</code></td>
<td>Store the selected object(s) in the paste buffer.</td>
</tr>
<tr>
<td><code>copy x y</code></td>
<td>Copy the selected object(s) and paste with reference point at ((\Delta x, \Delta y)).</td>
</tr>
<tr>
<td>Example:</td>
<td><code>copy 4 6</code></td>
</tr>
<tr>
<td><code>copy l x l y</code></td>
<td>Copy the selected object(s) and paste reference point at absolute location ((x, y)).</td>
</tr>
<tr>
<td>Example:</td>
<td><code>copy 4 10 -l Metal1 &lt;press Enter&gt; &lt;mouse click&gt;</code></td>
</tr>
</tbody>
</table>

**Goto**

Shift the screen display.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>goto x y</code></td>
<td>Shift screen center by ((\Delta x, \Delta y)).</td>
</tr>
<tr>
<td><code>goto !x !y</code></td>
<td>Move screen center to absolute coordinate ((x, y)).</td>
</tr>
<tr>
<td><code>goto -l x y</code></td>
<td>Move screen center to absolute coordinate ((x, y)).</td>
</tr>
</tbody>
</table>

**Instance**

Create an instance of the selected cell. If a cell name includes a space or comma, it must be enclosed in quotes.

Example:

```plaintext
instance "NAND 1" !14 !22 -f mainlib.tdb
```

creates a referenced instance of cell **NAND** from the **mainlib.tdb** file, with its origin at \((14, 22)\).

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>instance</code></td>
<td>Instance cell <strong>cName</strong>, placing the cell's origin at position (\Delta x, \Delta y) relative to screen center.</td>
</tr>
<tr>
<td>Example:</td>
<td><code>instance &quot;DFF R2&quot; 0 3</code></td>
</tr>
<tr>
<td><code>instance cName !x !y</code></td>
<td>Instance cell <strong>cName</strong>, placing the cell's origin at position (x, y).</td>
</tr>
</tbody>
</table>
Example: \texttt{instance DFF 42 136}

\texttt{instance} \texttt{cName -! x y} \hspace{1cm} \text{Instance cell} \texttt{cName}, placing the cell's origin at position \(x, y\).

Example: \texttt{instance DFF -l 42 136}

\texttt{instance} \texttt{cName <press Enter> <mouse click>} \hspace{1cm} \text{Instance cell} \texttt{cName}, placing the cell's origin at the position given by the mouse click.

**Layer**

Change the active layer to the layer designated using either the layer name or GDSII layer number.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{layer LayNum}</td>
<td>Change active layer to the designated GDSII layer number.</td>
</tr>
</tbody>
</table>

Example: \texttt{layer 42}

| \texttt{layer LayName} | Change active layer to the designated layer name. |

Example: \texttt{layer "SubCkt ID"}

**Move**

Move the selected object(s). The option \texttt{-R} followed by two relative coordinates specifies an offset of the reference point from the lower left corner of the minimum bounding box.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{move x y}</td>
<td>Move selected object(s) by (\Delta x, \Delta y).</td>
</tr>
</tbody>
</table>

Example: \texttt{move -40 -32}

| \texttt{move !x !y} | Move the lower left corner of the selected object(s) to \((x, y)\). If multiple objects are selected, this command prompts for a mouse click to determine the reference point for the translation to \((x, y)\). |

Example: \texttt{move !42 !136 <press Enter> <mouse click>}

| \texttt{move -! x y} | Move the lower left corner of the selected object(s) to the absolute location \((x, y)\). If multiple objects are selected, this command prompts for a mouse click to determine the reference point for the translation to \((x, y)\). |

Example: \texttt{move -! 42 136}

**Path**

Draw a wire between mouse click points in the layout or absolute coordinate points. The option \texttt{-l} changes the designated layer. The option \texttt{-pw} changes the wire width.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{path &lt;mouse click(s)&gt; &lt;right mouse click&gt;}</td>
<td>Draw a wire segment between mouse click points. Drawing is ended with a right mouse click. Note that wires are drawn in outline mode until the drawing is complete.</td>
</tr>
</tbody>
</table>
Chapter 9: Editing Objects

Editing

Example: path -l Metal1 -pw 12 <mouse click> <mouse click> <right mouse click>

**path** !x !y <mouse click(s)> <right mouse click>

Draw a wire from absolute location (!x, !y) to selected mouse click points, ending with a right mouse click.

Example: path !1092 !476 <mouse click> <mouse click> <right mouse click>

**path** !x1 !y1 !x2 !y2 ...

Draw a wire with vertices at the absolute locations (!x1, !y1) entered.

Example: path !-12 !14 !79.5 !16 !80.5 !9.5 !50 !-4.5 !23.5 !-15.5 !48.5 !-22.5

**path** -! x1 y1 x2 y2 ...

Draw a wire with vertices at the absolute locations (!x1, !y1) entered.

Example: path -! 4 10 11 15 20 20 30 45 45 60

**Paste**

Paste the selected object(s) into the layout. The option -l changes the designated layer.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>paste</strong> x  y</td>
<td>Paste the selected object(s), translating the reference point of the objects in the paste buffer by (Δx, Δy). Example: <strong>paste</strong> 15 25</td>
</tr>
<tr>
<td><strong>paste</strong> !x !y</td>
<td>Paste the selected object(s), translating the reference point of the objects in the paste buffer to the absolute location (x, y). Example: <strong>paste</strong> !1092 !476</td>
</tr>
<tr>
<td><strong>paste</strong> -! x y</td>
<td>Paste the selected object(s), translating the reference point of the objects in the paste buffer to the absolute location (x, y). Example: <strong>paste</strong> -! -12 -14</td>
</tr>
<tr>
<td><strong>paste</strong> &lt;mouse click&gt;</td>
<td>Paste the selected object(s) with the reference point of the objects in the paste buffer to the mouse click point. Example: <strong>paste</strong> &lt;press Enter&gt; &lt;mouse click&gt;</td>
</tr>
</tbody>
</table>

**Polygon**

Draw an all-angle polygon at the indicated absolute vertices. The option -l changes the designated layer.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>polygon</strong> !x1 !y1 !x2 !y2 ...</td>
<td>Draw a polygon with vertices at the indicated absolute coordinates. Example: <strong>polygon</strong> 15 25</td>
</tr>
<tr>
<td><strong>polygon</strong> -! x1 y1 x2 y2 ...</td>
<td>Draw a polygon with vertices at the indicated absolute coordinates. Example: <strong>polygon</strong> -! -12 -14</td>
</tr>
<tr>
<td><strong>polygon</strong> !x1 !y1 &lt;mouse click&gt;</td>
<td>Draw a polygon with first vertex at (x1, y1) and additional vertices indicated by mouse clicks. A right mouse click completes the command. Example: <strong>polygon</strong> !1092 !476 &lt;mouse click&gt; &lt;mouse click&gt; &lt;right mouse click&gt;</td>
</tr>
</tbody>
</table>
**polygon** <mouse click>  
Draw a polygon with all vertices indicated by mouse clicks. A right mouse click completes the command.

Example: **polygon** <mouse click> <mouse click> <right mouse click>

**Rotate**

Rotate the selected object(s) counterclockwise across an angular distance (in degrees) with respect to the specified point. The value of `angle` must be between -360 and +360 degrees, exclusively.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rotate angle x y</td>
<td>Rotate the selection counter-clockwise by <code>angle</code> (degrees) with respect to the point offset from the selection’s center by (Δx, Δy). Example: <strong>rotate 90 0 0</strong></td>
</tr>
<tr>
<td>rotate angle !x !y</td>
<td>Rotate the selection counter-clockwise by the <code>angle</code> (degrees) with respect to the absolute coordinates (x, y). Example: <strong>rotate 120 1092 476</strong></td>
</tr>
<tr>
<td>rotate angle -! x y</td>
<td>Rotate the selection counter-clockwise by <code>angle</code> (degrees) with respect to the absolute coordinates (x, y). Example: <strong>rotate 45 -12 26</strong></td>
</tr>
<tr>
<td>rotate angle &lt;mouse click&gt;</td>
<td>Rotate the selection counter-clockwise by <code>angle</code> (degrees) with respect to the point indicated by mouse click. Example: <strong>rotate 240 &lt;press Enter&gt; &lt;mouse click&gt;</strong></td>
</tr>
</tbody>
</table>

**Run**

Execute the sequence of commands in the specified file.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>run filename</td>
<td>run repwire.tco</td>
</tr>
</tbody>
</table>

**Text**

Create a point port at the specified location. The option -l changes the designated layer.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text label !x !y</td>
<td>Create a port at location (x, y) with the text string <code>label</code>. Example: <strong>text Gnd 14 110 -l Metal1</strong></td>
</tr>
<tr>
<td>text label -!x !y</td>
<td>Create a port at location (x, y) with the text string <code>label</code>. Example: <strong>text Gnd -1 4 10 -l Metal1</strong></td>
</tr>
<tr>
<td>text label &lt;mouse click&gt;</td>
<td>Create a port at the mouse click position with the text string <code>label</code>. Example: <strong>text Gnd &lt;press Enter&gt; &lt;mouse click&gt;</strong></td>
</tr>
</tbody>
</table>
Chapter 9: Editing Objects

**Width**

Set the wire width (in Display Units).

<table>
<thead>
<tr>
<th>Argument</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>Sets the wire width to the default value for the active layer. Example: <strong>width</strong></td>
</tr>
<tr>
<td>width wire width</td>
<td>Sets the wire width to the specified value. Example: <strong>width 3</strong></td>
</tr>
</tbody>
</table>

**Command Scripting**

Command scripting is supported through the use of a **run** command that opens and executes a text file containing a list of commands. The file format for this command list is Tanner Command Files with a **.tco** extension.

You can use the **Copy to file** command (**Ctrl+L** from within the window) to save highlighted text from the command window directly to a **.tco** script file.

C++ style comments are supported in command script files. The **run** command cannot be nested. Note that commands using mouse completion are not generally supported in command scripts.

For GDSII, allowable combinations of end and join styles are shown in the table below. All other combinations of end and join styles will produce an error message when exporting to a GDSII file.

<table>
<thead>
<tr>
<th>End style</th>
<th>Join style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butt</td>
<td>Layout</td>
</tr>
<tr>
<td>Round</td>
<td>Round</td>
</tr>
<tr>
<td>Extend</td>
<td>Layout</td>
</tr>
</tbody>
</table>

After you draw an individual wire you can change its individual style with **Edit > Edit Objects.**
10 Parameterized Cells

L-Edit has several processes for generating parameterized cells or device layout.

Parameterized cells are cells generated by layout generation code based on a set of parameters that can include electrical parameters, layer configurations or design rules. The default set of parameters in the master or parent cell can be overwritten when you generate instances of that T-Cell.

Tanner L-Edit parameterized cells are called T-Cells and use UPI code written in TCL, C, or C++. Because L-Edit’s UPI is so powerful, T-Cells can be very complex.

L-Edit also provides a callback feature that can check the validity of T-Cell parameters or modify other values when a related value is changed.

P-Cells

The OpenAccess database supports Cadence Virtuoso parameterized cells, called P-Cells, which use code written in the Skill language.

The default parameter values of a P-Cell may viewed and edited by opening the master cell, invoking Cell > Info, and selecting the T-Cell Parameters tab. L-Edit will preserve, read and save P-cell parameters, but cannot execute them.

The parameters of an instance of a P-Cell may be viewed and edited by selecting the instance, then invoking Edit > Edit Object (Ctrl+E) and selecting the T-Cell Parameters tab.

Since P-cell instances are not cached in OpenAccess and L-Edit cannot execute the Skill code to generate the P-cell with the specific instance parameter values, instances of P-Cells are displayed in layout as a simple placeholder (an empty cell with a white X through it.)

The L-Edit T-Cell Builder lets you automatically create simple T-Cell code views from layout geometry. The resulting T-Cells are parameterized and can be modified with these parameters.

Creating T-Cells

To create a new T-Cell, use Cell > New or to Cell > Info to enter parameters in the T-Cell Parameters tab. The T-Cell Parameters tab contains a table in which you can list parameters as input to generate T-Cell instances. For example, parameters for a decoder generator cell might include a user-specified number of outputs, number of bits, and pitch, and presence or absence of spacers.

You can also create a T-Cell by right-clicking on a cell in the Design Navigator and choosing Create/Update T-Cell Code. If the cell already has T-Cell code, L-Edit will display a warning and ask if you want to replace the existing code.
Use this pull-down menu to select the programming language of the T-Cell, from "TCL", "C", "C++" or "DLL". L-Edit opens the appropriate fields in template below.

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Prompt</td>
<td>Type</td>
<td>Default Value</td>
<td>Choices</td>
<td>Callback</td>
<td>Use</td>
<td>Query</td>
</tr>
<tr>
<td>L</td>
<td>length</td>
<td>Double</td>
<td>0.75u</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>width</td>
<td>Double</td>
<td>0.15u</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fingers</td>
<td>fingers</td>
<td>Cyclic</td>
<td>series</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>justify</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Boolean type parameters are presented as a check box in the Instance and Edit dialogs. Radio type parameters are presented as radio buttons in the Instance and Edit dialogs. Cyclic type parameters are presented as a drop-down list in the Instance and Edit dialog.

See also “H-Stretch and VStretch Parameter Types for Edge Selection,” below.

Default Value

Enter the default value of the parameter. For Boolean and Layer data types, a drop-down list of possible entries is provided. Parameters are case-insensitive.

Choices

Choices is a semicolon separated list of values for Radio and Cyclic parameter types. Do not put a space between items in the list.

Callback

Callback is the name of a TCL function to call when the value of a parameter is changed.

Use

A TCL expression that indicates that the property is used in the design. Default value is True if field is blank.
The **HStretch** and **VStretch** types set cell behavior when you select and drag a cell edge.

The default behavior when you select and drag an edge is to create an array by replicating the T-Cell instance, as shown below.

---

**Query**

A TCL expression that sets the display of the parameter in the Instancing and Editing dialogs. Only properties whose Use and Query values evaluate to True will be displayed in the parameter list when an instance of the cell is placed or modified. Default value is True if field is blank.

**Editable**

A TCL expression that sets the editability of the parameter. Only parameters on instances whose Editable value evaluates to True will be editable when an instance of the cell is placed or modified. Default value is True if field is blank.
However, when you set the parameter type to **HStretch** or **VStretch**, the behavior when you select and drag an edge is to extend the dimension of that parameter, as shown below.

---

**T-Cell Code Templates**

After you have finished entering parameters in the **T-Cell Parameters** tab, click **OK**. L-Edit will generate a template that is the skeleton of your T-Cell code.

This T-Cell template contains the outline for a UPI macro based on the parameters you’ve entered. Follow the directions in the comments to complete the code. You can encode your T-Cell with additional UPI functions if desired. T-Cell are precompiled as DLLs, and the L-Edit compiler supports C++.

Note that prior to version 16.00, dlls would be reloaded every time a T-Cell was invoked, which would reset all static variables. As of version 16.00, for performance reasons, L-Edit keeps dlls loaded in the .tdb file, so you may need to initialize certain parameters inside the generator.
The precompiler will delay performance the first time the code runs, but subsequent cell placement will be much faster.

```c
module mosfet_code
{
    #include <stdlib.h>
    #include <math.h>
    #include <string.h>
    #include <stdio.h>
    #include "ldata.h"

    /* Begin -- Remove this block if you are not using L-Comp. */
    #include "lcomp.h"
    /* End */

    /* TODO: Put local functions here. */
    void mosfet_main(void)
    {
        ...
    }

    /* TODO: Put local variables here. */
    /* TODO: Begin custom generator code.*/
    /* End custom generator code.*/
}
}
mosfet_main();
```
TCL Commands Specific to L-Edit

Type "help" in the Command window to see the following list of product-specific commands, with a brief description of what they do. Help for each of these can be expanded to display subcommands by typing the command name followed by "help," for example "database -help".

database -help
Help screen for command: database
Subcommands:
  open: open a tdb file
  new: open a new tdb file
  save: save a tdb file
  saveas: saveas a tdb file
  close: close a tdb file
  ischanged: report if a tdb file has unsaved changes
  cells: get the list of cells
  layers: get the list of layers
  instances: get the list of in a cell
  drcrules: get the list of drc rules in a design
  grids: get the grids in a design
Optional flags:
  -help: Display this help screen

L-Edit supports the following TCL commands.

barcode: create barcode from text
box: create a box
cell: open a cell (create if necessary)
circle: create a circle
cross: create a cross
database: manage and query the database
delete: delete an instance, or the current selection
drc: run drc on current cell
dtos: double to string
exit: exit command
findinstance: find matching instances
findoverlap: find overlapping cells
generatelayer: generate derived layer(s)
help: Display this help screen
hershey: create geometry from text
instance: create an instance or an array
locate: find the location of a named point
logmessage: write messages to log window
merge: merge objects on specified layers (or selection)
mgc_rve_context: specify cell for highlighting data
mgc_rve_export_layout: export layout
mgc_rve_get_location: input point
mgc_rve_get_rectangle: drag to select box
mgc_rve_highlight_index: set index of highlight color
mgc_rve_import: import layout
mgc_rve_layer: set highlight layer
mgc_rve_rect: highlight rectangle
mgc_rve_text: text on error layer
mgc_rve_text_clear: delete text on error layer
mgc_rve_zoom: zoom to box
moveorigin: relocates the origin of a cell
mr: highlight rectangle
pastetolayer: paste selection to layer
polygon: create a polygon
Opening T-Cells

T-Cells are shown with a generator icon ( ) next to them in cell lists. Note that Auto-generated cells are automatically hidden from the Design Navigator and other cell lists. To show auto-generated cells, select Show All Cells in the Design Navigator.

To open a T-Cell, choose Cell > Open (shortcut O) and select T-Cell Code in the Open view menu. You can also open a T-Cell by right-clicking in the Design Navigator and choosing Create/Update T-Cell Code. L-Edit will open a text window containing the T-Cell code for the cell, or a blank code template if the selected cell is not already a generator cell.

If an instance is selected in the layout, its name will be highlighted in the Open Cell dialog. If multiple instances are selected, the referenced cell of the first instance in the selection will be highlighted. If no instance is selected, the last cell opened will be highlighted.
Closing T-Cells

The **Close Cell As** dialog lets you save a T-Cell with its current name or enter a new name for the cell.

![Close Cell As dialog](image1.png)

When you use **Cell > Close As** to copy a T-Cell, all changes to the T-Cell code view are preserved in both the original cell and the copy, regardless of when the last save operation occurred.

![L-Edit Warning](image2.png)

**Warning:** Changes made to a T-Cell instance using **Edit > Edit In-Place** will be lost if you regenerate the T-Cell. The **Revert Cell** command does not reverse changes to T-Cell code.

Instancing T-Cells

An *instance* is a representation of a cell in a particular location and orientation in another cell. An instance can reference a cell composed of primitives, other instances, a combination of primitives and instances or a cell generated from T-Cell code (also referred to as a *generated cell*).

A T-Cell does not have geometry until it is instanced.

A quick way to instance a T-Cell is to click and drag it from the Design Navigator into a new cell. When you instance a T-Cell for which parameters have been defined, L-Edit opens the **T-Cell Parameters** dialog, where you can accept or edit the existing parameter values.

When instancing a T-Cell, L-Edit first generates a new cell containing the geometry specified by the T-Cell. This cell, called an *auto-generated* cell, is the source cell for the instance. The auto-generated cell is named by appending the word *Auto* followed by a number to the original T-Cell name, where the number is incremented for each new unique set of parameters. For example, the first time the dialog shown below creates an instance of an auto-generated cell, it will be named `MatchedDualCapacitorArray_Auto1`. 

![Auto-generated cell](image3.png)
Changes to a T-Cell, which L-Edit uses to generate source cells, cause all auto-generated cells and their instances to be flagged “out of date.”

You can update T-Cell instances using the **Cells > Regenerate T-Cells** command (see “Regenerating T-Cell Instances,” below).

---

**Regenerating T-Cell Instances**

Geometry in a T-Cell instance is generated by referencing an original T-Cell that contains a specific set of parameters. When either the original or an instance of a T-Cell is edited, it must be **regenerated** for the changes to take effect.

If you edit a T-Cell instance using **Edit > Edit In-Place**, the changes are propagated, but only to other instances that were created with identical parameter values. To edit the contents of *all* instances of a T-Cell, you must open the original T-Cell, make the desired changes there, and then regenerate cell instances to propagate the changes.

When you save changes to a source T-Cell, instances of that T-Cell will be flagged “out-of-date.” You use **Cell > Regenerate T-Cells** to propagate your changes in an original T-Cell to its instances.
The top of the **Regenerate T-Cells** dialog indicates whether the most recently edited source T-Cell, or cells generated from the most recently viewed T-Cell, are current. To update T-Cell instances, select one of the following options:

- **All**
  - Replaces all T-Cell instances in the active file with new instances that reflect the most current version of each T-Cell. Instances are re-created with the same parameters they originally used.

- **All out-of-date**
  - Updates only those T-Cell that are flagged “out-of-date.” The number in brackets indicates how many cells will be updated.

- **Cell TCell_Name**
  - Updates only the most recently edited T-Cell.

- **All cells generated from TCell_Name**
  - Updates only those instances that refer to the most recently active T-Cell.

When **Confirm each cell** is checked, L-Edit will prompt you to confirm the update of each T-Cell auto-generated cell.

**Note:** Because regenerating T-Cell instances replaces all auto-generated cells, any changes made to auto-generated source cells using **Edit > Edit In-Place** will be lost.

**T-Cell Callbacks**

The UPI function **“LCell_GetTCellPreviousValue”** (page 1310) lets you retrieve the previous value of a T-Cell parameter, which is the value used the last time the T-Cell was created or modified. This function has an identical signature to **LCell_SetTCellDefaultValue**, which can be used in conjunction
with \texttt{LCell\_GetTCellPreviousValue} to implement “callbacks,” by which a T-Cell can validate and modify the parameters that are passed to it.

\texttt{LCell\_GetTCellPreviousValue} is used to obtain the previous value of a T-Cell parameter the last time the T-Cell was created or modified. If the T-Cell is newly created, an empty string is returned.

The following examples show how \texttt{LCell\_GetTCellPreviousValue} can be used to determine whether a particular parameter has changed since the previous invocation of the T-Cell.

```c
int HasChanged( LCell pCell, char *param_name )
{
    char old_val[1024], new_val[1024];
    LCell_GetTCellPreviousValue(pCell, param_name, old_val, sizeof(old_val));
    LCell_GetTCellDefaultValue(pCell, param_name, new_val, sizeof(new_val));
    return strcmp( old_val, new_val );
}
```

Then, in the main T-Cell code body, we can use this information to modify other parameters. For example, suppose we had a resistor that was parameterized by \( R, L \) and \( W \). The values of these three parameters need to be consistent, so we need to modify one of them to enforce this consistency. One possible solution would be the following:

```c
// update new parameters accordingly
char new_val[1024];
if ( HasChanged(cellCurrent, "L") && HasChanged(cellCurrent, "W") )
{
    R = L / W * resistivity;
    sprintf( new_val, "%g", R );
    LCell_SetTCellDefaultValue(cellCurrent, "R", new_val);
}
else if ( HasChanged(cellCurrent, "L") )
{
    W = L * resistivity / R;
    sprintf( new_val, "%g", W );
    LCell_SetTCellDefaultValue(cellCurrent, "W", new_val);
}
else
{
    L = W * R / resistivity;
    sprintf( new_val, "%g", L );
    LCell_SetTCellDefaultValue(cellCurrent, "L", new_val);
}
```

---

**Generating T-Cell Code from Layout —“T-Cell Builder”**

The T-Cell Builder, accessed from the \texttt{Cell > T-Cell} menu, allows you to automatically generate T-Cell code views from existing layout views.

The resulting T-Cells are parameterized and contain geometry elements whose appearance can depend on these parameters. Because you do not need to write any code directly, this feature is very useful if you are not familiar with UPI programming.

A cell must contain \textit{stretch ports} in its layout to trigger the T-Cell builder. A T-Cell code view is constructed when you indicate the layer on which the stretch ports are drawn and analyze the geometry of the cell by executing the \texttt{Cell > T-Cell > Construct T-Cell} command. You can then keep or overwrite the parameters in that code view.
Once the cell has a code view, you can create parameterized layout by instancing it like any other T-Cell. Each time you add or change a parameter you must execute Cell > T-Cell > Construct T-Cell to apply the changes.

The T-Cell Builder can create parameters to either stretch (the default), move or repeat geometry according to the values you enter. The T-Cell Builder can also construct a parameter that sets the layer on which geometry is drawn, and a parameter for simple TRUE/FALSE conditions by which geometry is included when the T-Cell is instanced.

### Defining Stretch Ports for the T-Cell Builder

The stretch axis parameters are defined by special stretch ports on a user-selected layer. These ports define the name of the parameter that controls the stretch, the direction of the stretch, and the default value of the stretch. These ports may be drawn on any layer, but you must choose that layer as the stretch port layer when you construct your T-Cell.

Stretch ports must be line ports (i.e. have exactly one dimension and be non-zero). Stretch ports control stretch perpendicular to the direction in which they are drawn. In T-Cell builder processing, the port line is extended to the cell perimeter in both directions and any object it intersects is stretched by the value indicated in the port text. Objects that are entirely on one side of the port may be moved, depending on the direction of the stretch.

The default value of the parameter is taken from the size of the port, or by declaring it in the port string, in the form “"parameterName=defaultValue."”

When stretch ports are incorrectly defined or not defined at all, L-Edit displays this message:

![Message](image)

### Constructing a T-Cell with Stretch Parameters—MOSFET Example

A commonly-desired operation is for one or more dimensions of an object to be set by a cell parameter. For example, a simple MOSFET is often parameterized by its channel length and width. To create a parameterized MOSFET T-Cell, you first create the cell layout shown below.

Note the one-dimensional ports “length=2” and “width=5.” These ports, which can be on any layer you select, define the axes with respect to which cell elements will be stretched.
Stretching is the default operation if no other is specified.

The one-dimensional ports are the reference points for stretch operations.

Once you have the raw geometry and ports in a cell, you can invoke Cell > T-Cell > Construct T-Cell to generate T-Cell code from that layout.

L-Edit will prompt you to enter the Stretch port layer, which must be the layer on which the stretch ports are drawn. If there is preexisting T-Cell code, L-Edit will also confirm that you want to overwrite it.
After L-Edit constructs the T-Cell code you can instance the MOSFET into layout using the conventional **Cell > Instance** command, and change the values of the parameters as desired.

### Parameter Types in the T-Cell Builder

T-Cell builder parameters are characterized by certain types, as shown in the menu below. The **HStretch** and **VStretch** types are described in “H-Stretch and VStretch Parameter Types for Edge Selection” on page 376.
Repeating Elements with the T-Cell Builder

To repeat rather than stretch elements in a constructed T-Cell, first select the layout objects you want to repeat, then invoke **Cell > T-Cell > Define Repeat Group.**

Returning to the MOSFET example, you would select the four contact boxes, use **Cell > T-Cell > Define Repeat Group** to enter the repeat parameters shown in the **Define Repeat Group** dialog above, use...
Cell > T-Cell > Construct T-Cell to update and apply the parameters, and then instance the cell. Entering a width parameter value of 20 creates the **mosfet_Auto_20_2** cell shown below.

![Image of mosfet_Auto_20_2 cell](image)

**Setting the Layer as a T-Cell Builder Parameter**

The layer on which an object is placed can be made into a parameter of the T-Cell by selecting the object and invoking Cell > T-Cell Builder > Choose Layer. Only the objects that are selected when you open the Choose Layer dialog will have their layer parameterized.

![Choose Layer dialog](image)

<table>
<thead>
<tr>
<th>T-Cell parameter</th>
<th>Enter a name for the layer parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default layer</td>
<td>Pick a default layer for the choose layer parameter.</td>
</tr>
</tbody>
</table>

For example, in the resistor shown below, one segment is selected for **Choose Layer** parameterization.
Once the T-Cell is (re)constructed you can instance cell res and choose the layer on which the segment is drawn.

Defining Conditional Inclusion as a T-Cell Builder Parameter

You can use Cell > T-Cell Builder > Define Conditional Inclusion in the T-Cell builder to define parameters for an object such that it will be included or excluded based on a Boolean or logical condition.

A conditional exclusion can also be based on a logical expression if the expression uses a parameter that has been previously defined by ports on the layout. If the condition is true, then the object is included. For example, if you have a stretch port width you can use the logical expression width > 10. (In this case the default value is not relevant.)

For following example, in cell MOSFET select the box on layer HV Oxide, define a condition parameter called HighVoltage, and regenerate the T-Cell.

**Condition for Inclusion** Enter a name for the condition or an expression that uses defined parameters.

**Default value** Select FALSE or TRUE as a default value.
When you instance MOSFET and select value FALSE for parameter HighVoltage, the selected box will not be drawn.

Finding Objects that have T-Cell Builder Parameters

Cell > T-Cell Builder > Select T-Cell objects selects all the objects in a builder T-Cell for which a repeat group, conditional inclusion or layer parameter is defined. It is a useful shortcut since there is no visual indication of the objects in a T-Cell that have parameters.

Generating Via Cells

For compatibility with the OpenAccess database, vias in L-Edit are not just pure geometry but an object type with a definition that can be instanced. There are two types of via templates, standard and custom.

All via definitions are included in the cell list of the Libraries navigator. However, only those listed in the Setup > Design > Valid Vias list are considered valid vias for the active design and technology (see “Valid Vias” on page 396). When you define a new via of either type it is automatically added to this list, but you can also explicitly add vias as well.

Standard Vias

A standard via is defined by a name and a set of parameters. It is designated in the cell and other lists by a blue via icon [ ]. In an OpenAccess database, the parameters are stored in a “via def” in the technology file, and the standard vias are generated when you load a design or when a Virtuoso tech file that contains vias is loaded.
When you open or place a standard via, L-Edit reads the definition and creates a cell with the corresponding geometry. In a TDB file, the “Stdvia” definition is stored in the cell.

Custom Vias

A custom via is merely a reference to another via cell. For a custom via, designated by a red via icon [ ] in the cell list, all that is stored is the name. It may be an ordinary cell that contains just polygons or it can be a T-Cell or a P-Cell.

Via Icons

Newly defined vias are, by default, locked. Because the default is the locked state, in order to reduce clutter in the graphic interface, the icon for a locked via does not show a padlock. Instead, there is a special “unlocked” icon for vias, which is an open padlock.

icon for a **locked** via

icon for an **unlocked** via

You can globally set whether or not vias of any type can be selected using the **Edit > Allow Via Selection** toggle.

An icon with a red X indicates a via that **cannot be selected**.
Defining a Standard Via

Start from a blank new cell and use Setup > Define Standard Via Template. By default, L-Edit will add any new standard via cell definition to the Setup > Design > Valid Vias list.

**Define Standard Via Template**

- **Via name**: Enter a name for the via cell.
- **Cut Layer**: Select the connector layer from the layer menu.
- **Cut Size**: Enter X- and Y- values of the connector layer.
- **Rows and Columns**: Enter integer values if you want to generate multiples of the cut geometry within a single via cell.
- **Pitch**: Sets the center to center X- and Y- distances between multiple connect cuts.

This setting determines the via spacing when an array of vias is created for a transition between layers on wide metal routes. It is also used to determine the minimum spacing of two vias on the same net.
### Origin Offset
Allows you to set an \(X\)- and \(Y\)-distance by which the center of the via will be offset from the cell origin. (The default offset is zero so the center of the via and the cell origin align.)

### Lower Layer, Upper Layer
Use the pull-down list to select the upper and lower via layers.

### Surround cut by
Use this field to set the dimensions of the lower and upper via layers with respect to the connect layer.

### Offset
You can use these fields to shift the position of the surround with respect to the center of the related layer.

### Implant Layer
Select any other layers on which to generate geometry (optional).

## Defining a Custom Via

A custom via is simply a pointer to a cell containing the geometry that will be instanced as a via. Only the name of this target and the layers it uses are stored.

To add a new custom via, draw the desired geometry in a new cell, then use **Setup > Define Custom Via Template** to give it a name and to specify a lower and an upper layer for automatic routing.

### Define Custom Via Template

- **Name**: Enter a name for the custom via.
- **Via Cell**: Select the cell the custom via points to.
- **Lower Layer, Upper Layer**: Use the pull-down list to select the upper and lower via layers.

**Note**: Regardless of the layer geometry of the referenced cell, L-Edit will use the upper and lower layers entered in this dialog for automatic and assisted routing.

- **Pitch**: Sets the center to center \(X\)- and \(Y\)-distances between multiple connect cuts.
Editing a Via

To edit a via definition, use Via Parameters in the Libraries navigator context menu to open the Define Standard Via Template or the Define Custom Via Template as appropriate.

To edit an instance of a via, select the instance and use Edit > Objects to open the Edit Object(s) dialog for vias, then click the Via parameters button to open the Edit Via dialog, where default values are displayed in blue text and non-default values are displayed in black text.

The blue arrow to the right of the Via Name field [ ] resets the x- and y-values in the active field to their defaults.

Placing a Generated Via

To instance a via, you can click on the Place Via icon in the drawing toolbar [ ], drag-and-drop from the Libraries navigator, or you can use the Draw > Vias > Place Via dialog, which provides a list of vias exclusively. You can further limit this via list with the checkbox to Show only valid vias, which are those included in the Setup Design Vias tab, and you can use the filter fields to show just certain via names, certain upper layers or certain lower layers.
The **Size** group box lets you create multiple rows and columns of the via cut layer as you place it. If the via definition already includes multiple rows or columns, **Place Vias** will override those values.

The **Place Via** dialog box is shown below:

---

**Valid Vias**

Only vias that are included in the **Valid Vias** list are available for use with the assisted routing shortcut keys. (The valid vias list is also used to initialize the via cell list in the **Setup Router** dialog in SDL.)

When you define a new via of either type it is automatically added to this list. However, you can explicitly add or remove vias from this list.

When you open an earlier version TDB file in version 16, L-Edit automatically creates a custom via definition for each cell in listed in **Setup > Design > Valid Vias**, and uses them to populate the version 16 Valid Vias list. The pitch will bw initialized with values from the earlier version TDB.
**Setup > Design > Valid Vias** lists the vias that are properly defined for use with assisted routing in the current design. Their order in this list is the order in which the assisted router will cycle through vias when you use the Cycle routing command. Note that the Layer and Pitch fields are read-only.

To add a via to this list, click the add button and then the browse button (…) in the **Via Name** field to open the **Pick Via** dialog.

**Filters**

If the via list is long, you can use the **Name**, **Lower layer** and **Lower layer** fields to limit the display to just the values you enter or choose.

You can right-click and use the Info menu item from this dialog to edit via pitch and layer values.

See “Standard Navigation Buttons in List Dialogs” on page 187 for a description of these buttons.
Using Valid Vias for Assisted Routing

L-Edit includes an assisted routing feature that uses a set of shortcut keys to simultaneously place a via and switch between the upper and lower routing layers used in that via.

L-Edit chooses vias for assisted routing based on their inclusion in the valid vias list, and the routing layers they use. (You may have other via cells in your design, but only the cells in this list can be used with the shortcut keys that speed routing.)

Routing Example

For example, let’s say you have three vias in your valid list—M1M2 with lower layer on metal1 and upper layer on metal2, M2M3 with lower layer on metal2 and upper layer on metal3 and M3M4 with lower layer on metal3 and upper layer on metal4.

When you use an assisted routing shortcut key, L-Edit will pick a via with its lower layer on the active layer if you press “up,” or a via with its upper layer on the active layer if you press “down,” place that via, and switch to the other layer the via uses, as shown in the following illustration.
How to Use Assisted Routing

With a routing layer active, invoke **Draw > Contacts** to open the menu shown below, and select one of the options to place a via on the current routing layer centered at the mouse location.

![Menu Screenshot]

Use **Draw > Contacts > Down (Single)** (shortcut key left bracket `[`) to instance a via cell whose upper layer is the current routing layer and to change the active layer to the lower routing layer of that via. Similarly, use **Draw > Contacts > Up (Single)** (shortcut key right bracket `]`) to instance a via having the active routing layer as its lower layer and continue routing on its upper routing layer.

You can use the `[` and `]` keys to initiate placement of a generated via with or without first placing a wire vertex.

When you are drawing with a wide wire, the `[` and `]` commands will automatically place an array using the pitch value for the center-to-center distance. (See “Placing an Array with Assisted Routing” on page 400.)

Editing a Via During Placement

If you do not move the cursor immediately after you place the via, the via remains selected and you can edit it. When you are in this ‘Wire Pending” mode, you can rotate, flip or stack the via with other vias. Once you have are done editing the via, move the cursor to draw wire on the new routing layer.

If you move the cursor (by more than half the width or height of the via) at any time after placing a generated via, L-Edit returns to Wire Drawing mode and starts a new wire on the new layer.

Cycling Through Multiple Vias with the Same Layer Definitions

If more than one via is defined that meets the routing layer criteria, you can use **Draw > Vias > Cycle** (shortcut key single apostrophe `') to cycle through those vias, in the order in which they are appear in the **Valid Vias** dialog.

When you come to the one you want, click to select and place it. If a via is already selected in the layout, the cycle function replaces it with the next via in the list that connects the same two layers, if one exists. Or, if you press `]` again, L-Edit will cycle to the next appropriate via but then stack it on top of the active via instead of replacing it.
‘Undo’ During Assisted Routing

L-Edit remains in Wire Pending mode until you place the next wire vertex. This allows L-Edit to retain routing layer data for the Undo function. **Undo** after placing a via will also revert so that the correct layer is active. (The correct layer is the one that was not active of the two layers associated with the via instance removed by undo.)

Placing an Array with Assisted Routing

You can also use the shortcut commands **Draw > Vias > Down (Area)** shortcut **Shift+[** and **Draw > Vias > Up (Area)** (shortcut **Shift+]**) to explicitly select an area inside of which to place a multi-cut via cell. Click on opposing corners to set the size of the contact array (see illustration below).

---

**Generating Guard Rings**

L-Edit can automatically generate an array of special via cells to create a guard ring that surrounds or covers layout objects.
Defining a Guard Ring

Use **Setup > Define Guard Ring Template** to enter parameters for the via cell that will be placed in the guard ring.

- **Cell name**: Enter a name for the via cell.
- **Library**: Select a library where the via cell will be stored.
- **Via Layer**: Select the connector layer from the layer menu.
- **Via Size**: Enter X- and Y- values of the connect layer.
- **Pitch**: Sets the minimum center to center X- and Y- distances between two contacts.
- **Lower Layer, Upper Layer**: Use the pull-down list to select the upper and lower via layers.
- **Surround via by**: Use this field to set the dimensions of the lower and upper via layers with respect to the connect layer.
- **Fracture**: Check **Fracture** to generate separate geometry for each individual via (see “Using the Fracture Option,” below).
Generating Guard Rings

Chapter 10: Parameterized Cells

Using the Fracture Option

When L-Edit generates a guard ring, the default is to merge via geometry on every layer.

For any layer but the via layer, you can check Fracture to generate separate geometry on that particular layer when you place the cell.

With Fracture checked, each box on each layer of the via is separate.

When the fracture option is not checked, geometry from each via is merged on each layer (default).
Using the Area Option

When Area is checked, L-Edit will generate guard ring geometry that covers a Boolean OR union of the selected geometry and adds the additional Surround via by area you have specified.

Editing a Guard Ring Cell

To edit an existing guard ring cell, left-click on the guard ring cell in the cell list and click on the menu item Guard Ring Parameters to open the “Define Guard Ring Template” dialog. You can change values independently or use the Remove Guard ring Info button to delete all values but the cell’s name.

Placing a Guard Ring

L-Edit will automatically create a guard ring around the selected geometry when you use one of the Draw > Guard Rings commands. There are four options. For each of the guard ring options except delete, you enter the following information.

Guard ring cell
Enter the name of the guard ring cell that will comprise the guard ring.

Library
Select the library where the guard ring cell will be saved.

Spacing to selection
Enter the orthogonal distance from the selected geometry to the inner edge of the guard ring.
In the following illustrations, all the objects are selected.

**Draw Guard Ring Around Selection**

The guard ring is drawn around the selected objects. When wires are selected, the guard ring follows the contour of the wire.

**Draw Guard Ring Around MBB of Selection**

The guard ring around is drawn around the minimum bounding box of the selected objects.
Because L-Edit recognizes its own generated cells, when you select an entire cell and its guard ring, you can use Draw > Guard Rings > Delete to delete just the guard ring cells and no other geometry.

After using Draw > Guard Rings > Delete, the cell remains intact—only the guard ring geometry is deleted.
11 Automatic Device Generation

DevGen™ and HiPer DevGen™ are tools that accelerate analog layout by generating fully parameterized device layout for analog devices and analog device primitives in L-Edit. They build on Tanner EDA’s existing parameterized T-Cells so that once a device is configured you can instantly update its parameters to regenerate entirely new devices.

You can specify the process definition, layout configurations, design rules, and electrical parameters for each device type, including MOS transistors and resistors as well as MOS current mirrors, differential pairs, resistor dividers and other basic structures.

DevGen T-Cells are stored and saved with the .tdb file.

HiPer DevGen advances standard DevGen in several ways. It can read matching, device and interconnect parasitics, performance considerations and silicon area requirements so that it generates layout that has these design goals built in. Foundry manufacturing rules and matching requirements are taken into consideration to produce specific structures tuned to your optimum solution, with minimal design time.

HiPer DevGen can also recognize schematic configurations that adhere to a given set of design constraints to produce device layout from a netlist using the schematic driven layout feature. If you use a netlist from Tanner’s S-Edit, the layout positions of generated devices will correspond to their position in the schematic.

The DevGen Process

Establish the Process Technology

Both DevGen and HiPer DevGen device generation are highly dependent on process technology. DevGen process setup is local to the cell and must be defined for each type of device.

Define the Device

Each device type has its own device definition wizard. Use **Cell > DevGen T-Cell > Define {device type}** to create or modify a set of device parameters, which will be saved as a T-Cell.

<table>
<thead>
<tr>
<th>Define Capacitor...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define Diode...</td>
</tr>
<tr>
<td>Define Inductor...</td>
</tr>
<tr>
<td>Define MOSFET...</td>
</tr>
<tr>
<td>Define Resistor...</td>
</tr>
<tr>
<td>Define HiPer Current Mirror...</td>
</tr>
<tr>
<td>Define HiPer Differential Pair...</td>
</tr>
<tr>
<td>Define HiPer MOSFET...</td>
</tr>
<tr>
<td>Define HiPer MOSFET Array...</td>
</tr>
<tr>
<td>Define HiPer Resistor...</td>
</tr>
<tr>
<td>Setup HiPer DevGen Process Information...</td>
</tr>
<tr>
<td>Edit DevGen Definition...</td>
</tr>
</tbody>
</table>
Generate Device Layout

Once you have created the default DevGen T-Cell you can generate specific instances of a device by clicking and dragging it from the Design Navigator. This will open a “Generate (DevGen device type)” dialog containing the same fields as the definition pages. You can accept the parameters as entered in the device definition wizard, or change them for the specific instance you are generating.

Edit a Device Class or Create a New One

Use Cell > DevGen T-Cell > Edit DevGen Definition to update the default DevGen cell definition or to create a new class of the same device that has different defaults. See “Editing a Device Class” on page 408.

The HiPer DevGen Process

Establish the Process Technology

To use HiPer DevGen in a layout database file, you must first establish the process technology by entering layer assignments, design rules and certain default parameters. HiPer DevGen process setup is global for all devices (see “HiPer DevGen Process Setup” on page 441). Each HiPer DevGen wizard page includes a Setup Process shortcut button so you can adapt the process and technology settings for a given cell definition.

Define the Device

Each device type has its own device definition wizard. Use Cell > DevGen T-Cell > Define HiPer {device type} to create or modify a set of device parameters, which will be saved as a T-Cell.

Generate Device Layout

Once you have created the default HiPer DevGen T-Cell you can generate specific instances of a device by clicking and dragging it from the Design Navigator. This will open a generate “HiPer {device type}” dialog containing the same fields as the definition pages. You can accept the parameters as entered in the device definition wizard, or change them for the specific instance you are generating.

Generate Device Layout from an SDL Netlist

HiPer DevGen works with the schematic-driven layout capability in L-Edit to read a netlist, recognize certain types of subcircuits, and generate layout for these device types. See “Device Generation From an SDL Netlist” on page 432.
Standard DevGen Devices

DevGen Capacitor

Use the **Cell > DevGen T-Cell > Define Capacitor** wizard pages to construct a single capacitor or a capacitor array with optional dimension controls that let you constrain one dimension to a fixed value or create a square capacitor.

The capacitor array generator builds large and precision capacitors from evenly distributed arrays of small and accurate unit capacitor elements. Capacitor arrays are widely used in demanding analog circuit implementations such as switch-cap circuitry in filter applications. The advantages of such structures include precision value and uniform parasitic conditions on unit capacitor elements.

DevGen Diode

Use **Cell > DevGen T-Cell > Define Diode** to construct a two-terminal device that conducts electric current in only one direction.

DevGen Inductor

Use **Cell > DevGen T-Cell > Define Inductor** to construct inductor cells in either rectangular or circular coils. Parameters include the width of inductor segments and the total number of wraps in the coil.

DevGen MOSFET

Use **Cell > DevGen T-Cell > Define MOSFET** to construct an n- or p-channel MOSFET cells using specified transistor parameters and layout options. An optional control allows you to constrain one dimension of the device to a fixed value.

DevGen Resistor

Use **Cell > DevGen T-Cell > Define Resistor** to construct continuous (simple) or segmented (precision) resistor cells. Dimension control allows you to specify the width and number of resistive layout segments.

Editing a Device Class

Each T-Cell device name contains a local copy of process information such as design rules and a saved set of default parameters. You can specify several different default configurations simply by modifying the T-Cell device name. Use **Cell > DevGen T-Cell > Edit DevGen Definition** to modify the existing T-Cell.
HiPer DevGen Devices

HiPer DevGen Current Mirror (page 409)

Constructs a circuit that copies a current through one active device by controlling the current in another active device to keep the output current constant regardless of loading.

HiPer DevGen Differential Pair (page 415)

Constructs a differential pair with multiple connection and matching options.

HiPer DevGen MOSFET (page 421)

Constructs an n- or p-channel MOSFET cell using specified transistor parameters and layout options.

HiPer DevGen MOSFET (page 421)

HiPer DevGen MOSFET Array (page 423)

HiPer DevGen Resistor (page 428)

Constructs continuous (simple) or segmented (precision) resistor cells and resistor arrays.

HiPer DevGen Current Mirror

The majority of parameters in the current mirror describe the actual reference device (diode device). These include width, length, number of fingers, well contact, multiplicity and supply rail width. The parameters set here are applied to all devices in the mirror.
Current Mirror Definition - Page 1

Use Cell > DevGen T-Cell > Define Current Mirror.

Device (T-Cell) name
Enter a name for the device you are creating.

NMOS device / PMOS device
Specify the type of MOS to be used in the mirror by choosing one of these radio buttons.

By convention, an NMOS usually has the ground rail below it, and a PMOS has the supply rail above it. So if a PMOS mirror is specified, the PMOS mirror will be vertically flipped with respect to an NMOS mirror.

Channel length
Specifies the channel length of the diode (unit) device. All other devices in the mirror will have the same unit channel length.

Channel width
Specifies the channel width of the diode (unit) device. All other devices in the mirror will have the same unit channel width.

Number of fingers
Specifies the number of fingers (gates) that the diode (unit) device is made from. All other devices in the mirror will have the same number of fingers.
Chapter 11: Automatic Device Generation

HiPer DevGen Devices

**Multiplicity**
Specifies the multiplication factor that should be applied to the diode (unit) device. The same factor is applied to all other devices in the mirror.

**Bulk contact location**
Select either option:
- **Under supply rail**

![Diagram showing bulk contact below supply rail]

The bulk contact (magenta) runs below the supply rail.

- **Under supply rail and between devices**

![Diagram showing bulk contact below and between devices]

The bulk contact (magenta) runs below and between devices.

**Supply rail width (power for PMOS, ground for NMOS)**
Sets the width of the supply rail (in metal1) that is to be used in the current mirror. The width will have an effect on IR drop across the rail, as well as current density/electromigration. The rail is below devices in an NMOS mirror and above them in a PMOS mirror.
Number of mirror units

Sets how many times the reference current is mirrored. The value must be any integer greater than or equal to 1.

This value specifies how many units there are in the mirror but does not include the diode device itself. In other words, it is the number of other unique devices in the mirror.

For each unit, there must be a corresponding entry in the List of unit scaling for each mirror and List of output current interconnect widths fields.
List of unit scaling for each mirror

The length of this list must equal the value specified in **Number of mirror units**.

For each mirrored device in this list, the number specified here is an integer scaling which is applied to the mirrored current/unit device. This scaling is applied depending on the combination of width, number of fingers and multiplicity specified for the diode device. For the best matching, the diode device should always be placed in the center of the current mirror. All other units should be placed symmetrically around this center. To guarantee the best matching, the unit multiplier for each device should be an even number.

List of output current interconnect widths

The length of this list must equal the value specified in **Number of mirror units**.

For each mirrored device in this list, the number specified here dictates the width of the track that carries the output/mirrored current. The width must always be wide enough to accommodate a via1. Depending on track width, HiPer DevGen many place multiple rows of vias.

Number of rows

Sets the number of rows in the current mirror. The **Multiplicity** parameter must be an integer multiple of this value, as each row in the current mirror array must contain at least one unit of the diode/reference device.

To ensure matching and uniformity of current flow through the current mirror array, the number of rows should always be even (unless it is a single row), and each row should contain at least one unit of each device in the mirror. In other words, **Unit Mult × Multiplicity** for each unit should be evenly divisible by this value. Each alternate row in the current mirror will be flipped.

Add dummy devices

For optimal matching, dummy devices should be placed at each end of each row in the current mirror. These dummy devices are made up of the same unit device that makes up the diode device. This ensures best matching, and also provides a contingency for metal mask spins.

The drain and source of the dummy are tied to the supply, and the gate is tied to the reference current, so it also acts as local decoupling of the reference current. In the case where devices in the mirror share diffusion, if dummies are added, the dummies will also share diffusion with adjacent devices.

For current mirrors comprising of more than one row, if dummies are added, dummies will be added to the ends of each row, so the total number of dummies is equal to (2 × number of rows).
In order to minimize area, you can try to share diffusion by overlapping adjacent devices in the current mirror. This is only possible if the number of fingers used in the diode device is even.

If the number of fingers is even, the common source connection will be on the outside of the device, allowing sharing. If the number of fingers is odd, the drain of the device is on the outside and as this is not a common connection between devices in the mirror, sharing will be disallowed.

The option to **Share diffusion between adjacent devices** will override the **WPE** option if that is enabled (checked), as it is not possible to share the devices if a well connection is between them.

The parameter **Match STI stress/LOD effects** also has an impact on whether sharing of diffusion is allowed. In most cases if **Match STI stress/LOD effects** is enabled, then sharing of diffusion is not allowed.

This applies to PMOS devices only. If this box is checked, the N-Well enclosure of P-Diffusion will obey the DRC rule. If it is not checked, HiPer DevGen adheres to normal N-Well enclosure of P-Diffusion.

In order to guarantee STI (shallow trench isolation) matching, the LOD (length of diffusion, also known as poly-to-active edge spacing) must be the same for all devices in the current mirror. If you do not try to share diffusion, this will always be the case.

However, if the devices are to be shared, then it is not always possible to match STI. This **Match STI stress/LOD effects** parameter has priority over the parameter **Share diffusion**. If it is possible to do both, HiPer DevGen will do so. However, when **Match STI** is enabled, it will override diffusion sharing if necessary.
Generate a Current Mirror

Once you have completed the current mirror definition you can generate an instance by dragging the cell name from the Design navigator into the layout to open the dialog shown below.

HiPer DevGen Differential Pair

Both devices in the differential pair should share the same type, width, length and number of fingers. They should also have a common source connection (typically tied to a current mirror). For this reason, you only need to specify the type, width, length and number of fingers for one device and HiPer DevGen will automatically assign these properties to the other device.
Differential Pair Definition - Page 1

Use **Cell > DevGen T-Cell > Define Differential Pair.**

**Device (T-Cell) name**: Enter a name for the device you are creating.

**NMOS device / PMOS device**: Specify the type of MOS to be used in the differential pair by choosing one of these radio buttons.

**Channel length**: Set the length of the channel of the transistors being used in the differential pair.

**Channel width**: Set the width of the channel of the transistors being used in the differential pair.

**Number of fingers**: The number of fingers (gates) used in each device of the differential pair.

**Note**: In order to create a matched structure, this number must be even.
**Drain wire width**

Where this is necessary, it is possible to specify the width of the metal used to join the drains together. The metal used for this connection is metal3. For every combination of differential pair, this connection is required, with the exception of a two-fingered differential pair where the drain parasitics are being optimized. (In this case the drains are shared, so no further interconnect is required.)

**Drain wire spacing**

You can specify the spacing between the input/gate connection on metal1 to the output/drain connection on metal3 in order to reduce parasitic coupling.

**Source wire width**

The sources in the differential pair are connected to the same net. This common connection is made via a metal3 route running uniformly across the middle of the device. By varying the width of the source wire you can vary the width of this track. Note that the maximum width of this track is limited by the width of the channel of the device.
Differential Pair Definition - Page 2

**HiPer Differential Pair Setup Wizard**

**Step 2/2 - HiPer Differential Pair Default Output Mirror Parameters**
Device Name: HiPer Differential Pair_1

- **Generation defaults**
  - [ ] Connect gate on top and bottom
  - [ ] Add guard ring
  - [ ] Adjust for well proximity effect (WPE)
  - [x] Add dummy devices: if necessary for matching
  - [ ] Add protection diodes

- **Matching requirements defaults**
  - [ ] Match drain parasitics
  - [ ] Match source parasitics
  - [ ] Match drain parasitics to source parasitics
  - [ ] Match linear process gradients

- [ ] Match STI/LDD effect
- [ ] Optimize area

- Optimize parasitics: Drain parasitics

- [ ] Setup Process

---

**Connect gate on top and bottom**
Check this box so the gate is connected on both top and bottom of the device. This will reduce the parasitic resistance in series with the gate.

**Add guard ring**
Check this box to surround the differential pair with a guard ring.

A guard ring will reduce well resistance, ensure the resistance of the well is distributed evenly across both devices in the differential pair, and can also provide some noise immunity or shielding.

**Adjust for well proximity effect (WPE)**
This applies to PMOS devices only. If this box is checked, the N-well enclosure of the device will be adjusted to obey WPE rules.

**Add protection diodes**
When this option is checked, HiPer DevGen adds a protection diode to each gate.

During fabrication, charge can build up on any gates tied to metal. This charge can cause a threshold voltage shift, and on matched structures, can lead to a mismatch. In order to reduce or match this shift, you can add antenna diodes to the gates of a device, connected to them in metal1.
Generate a Differential Pair

Once you have completed the differential pair definition you can generate an instance by dragging the cell name from the Design navigator into the layout to open the dialog shown below.

During generation, you can choose to prioritize whether to optimize the drain or the source parasitics. Note that in one particular case, it is also possible to ensure that the drain and source have identical parasitics to each other, so one is not prioritized over the other, in which case this option is ignored.
Optimizing parasitics means that the overall drain or source area is reduced by sharing or overlapping devices. This results in a smaller physical area leading to a reduced capacitive load and a reduced parasitic resistance in series with either the drain or source.

**Multiple Combinations of Matching Requirements**

Note that generation options are not mutually exclusive of each other—they can be used together in any combination. For example, a differential pair can have antenna diodes, a guard ring, gates connected on both sides and dummies included.

Depending on the number of fingers and on whether the designer wants to optimize drain over source, source over drain or indeed to match both drain to source, then only a number of possible solutions are available. Hence there are cases, where a certain combination of matching requirements will lead to a scenario where there is no solution. For this reason it is possible that a matching requirement will be adhered to even if it is not checked.

The matching requirements are considered to be a priority order. HiPer DevGen will attempt to produce the optimal matched layout, however if a priority flag is set, it will always adhere to that requirement first to the detriment of others.
HiPer DevGen MOSFET

MOSFET Definition

Use **Cell > DevGen T-Cell > Define MOSFET.**

---

**Device (T-Cell) name**

Enter a name for the device you are creating.

**NMOS device / PMOS device**

Specify the type of MOS to be used in the MOSFET by choosing one of these radio buttons.

By convention, an NMOS usually has the ground rail below it, and a PMOS has the supply rail above it. So if a PMOS MOSFET is specified, the PMOS MOSFET will be vertically flipped with respect to an NMOS MOSFET.

**Channel length**

Specifies the channel length of the device. This must be a multiple of the manufacturing grid.

**Channel width**

Specifies the channel width of each finger in the device. This must be a multiple of the manufacturing grid.

**Number of fingers**

Specifies the number of fingers in the device. The total unit width is equal to the width * number of fingers.
### Multiplicity

The multiplicity of the device. Note that this does not join all drains and sources together if gate connections and well connections are shared.

### Adjust for well proximity effect (WPE)

This applies to PMOS devices only. If this box is checked, the N-well enclosure of the device will be adjusted to obey WPE rules.

### Join multiple fingers

Check this box to join all source connections and join all drain connections in Metal 1 layer (when the number of fingers is greater than 1).

### Gate inter-connection

Sets the location of the contact for the gate, and whether you want to connect both the top and bottom gates together for multiple finger transistors.

Options are:
- None – Do not use a gate contact.
- Top – Place the gate contact on the top and connect all the gates of multiple finger transistors together with Poly layer on the top of the transistor.
- Bottom – Place the gate contact on the bottom and connect all the gates of multiple finger transistors together with Poly layer on the bottom of the transistor.
- Top & Bottom – Place the gate contacts on the top and bottom and connect all the gates of multiple finger transistors together with Poly layer on the top and bottom of the transistor. This will reduce the overall gate resistance and is good when you need low noise transistors.

### Bulk contact arrangement

Sets the location of the bulk connector.

Options are:
- None – Do not use a bulk connection.
- Top – Add a bulk connection to the top of the transistor.
- Bottom – Add a bulk connection to the bottom of the transistor.
- Top & Bottom – Add a bulk connection to the top and bottom of the transistor.
- Left – Add a bulk connection to the left of the transistor.
- Right – Add a bulk connection to the right of the transistor.
- Left & Right – Add a bulk connection to the left and right of the transistor.
- Surround – Add a guard ring for the bulk around the transistor.
Generate a MOSFET

Once you have completed the MOSFET definition you can generate an instance by dragging the cell name from the Design navigator into the layout to open the dialog shown below.

HiPer DevGen MOSFET Array

Note that the default length, width, number of fingers and multiplicity can be changed for each MOSFET array in each instance of the T-Cell.
MOSFET Array Definition

Use Cell > DevGen T-Cell > Define MOSFET Array.

Device (T-Cell) name
Enter a name for the array you are creating.

NMOS device / PMOS device
Choose either NMOS or PMOS array.

Channel length
Specifies the default channel length for each MOSFET added to the array. This must be a multiple of the manufacturing grid.

Channel width
Specifies the default channel width for each MOSFET added to the array. This must be a multiple of the manufacturing grid.

Number of fingers
The default number of fingers for each MOSFET added to the array.

Multiplicity
The default multiplicity for each MOSFET added to the array.

Number of rows
The default number of rows for the MOSFET array.

Track width
The default track width for each port of the MOSFET array. (A port is a net of the MOSFET array that is connected outside of the array.)
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Guard ring arrangement</strong></td>
<td>Sets the guard ring type for the MOSFET array. Options are:</td>
</tr>
<tr>
<td></td>
<td>← None – Does not add a guard ring.</td>
</tr>
<tr>
<td></td>
<td>← Device – Surrounds each device in the array with a guard ring.</td>
</tr>
<tr>
<td></td>
<td>← Row – Surrounds each row in the array with a guard ring.</td>
</tr>
<tr>
<td></td>
<td>← Mos Array – Surrounds the entire array with a guard ring.</td>
</tr>
<tr>
<td><strong>Rotate MOSFETs by 90 degrees</strong></td>
<td>Rotates the array by 90 degrees.</td>
</tr>
<tr>
<td><strong>Adjust for well proximity effect (WPE)</strong></td>
<td>Applies to PMOS devices only. If this box is checked, the N-well enclosure of the device will be adjusted to obey WPE rules.</td>
</tr>
</tbody>
</table>
Chapter 11: Automatic Device Generation

HiPer DevGen Devices

Generate a HiPer MOSFET Array

Once you have completed the MOSFET definition you can generate an instance by dragging the cell name from the Design navigator into the layout to open the dialog shown below.

Number of devices
Enter the number of MOSFETs in the array.

number of rows
Enter the number of rows in the array.

Guard ring arrangement
Sets the guard ring type for the MOSFET array. Options are:

← None – Does not add a guard ring.
← Device – Surrounds each device in the array with a guard ring.
← Row – Surrounds each row in the array with a guard ring.
← MOS Array – Surrounds the entire array with a guard ring.
Rotate MOSFETs by 90°

Rotates the array by 90 degrees.

Device Parameters

Sets the width, length, number of fingers, multiplicity and net names for the drain, gate and source.

Right-click on any field in this pane to add or delete a row.

By default, the width, length, number of fingers and multiplicity are initialized to the values from the MOSFET array definition.

Nets is a text field that will allow any valid SPICE node name.

Nets

This list of netnames is populated as you add net names in the device parameters area.

Netname is the name of a net in the MOSFET array.

Is Port – indicates whether the net is a port. (A port is a net of the MOSFET array that is connected outside of the MOSFET array.)

Note: You must have at least one port defined.

Track Width sets the width for horizontal and vertical tracks for each net.

Device arrangements

This chart lets you specify the relative location of each MOSFET in the MOSFET array. If you do not change the default settings, L-Edit will use a “non-matched” simple pattern list.

When you press Initialize L-Edit will insert the proper fields with their default values. Each device will be color coded. You can right-click anywhere in this pane to add or delete a row, but you may not have an empty row or a discrepancy with respect to the other device parameters.

To adjust the arrangement, double-click inside a color block to make it editable, and change the device number.
HiPer DevGen Resistor

Resistor Page 1

Device (T-Cell) name
Enter a name for the device you are creating.

Resistor type
Select from the options N+ Poly, P+ Poly, N+ Unsalicided Poly, P+ Unsalicided Poly or N-Well.

Length
Specifies the default channel length of the device. This should be a multiple of the manufacturing grid, otherwise the T-Cell will snap to the manufacturing grid anyway.

Width
Specifies the default channel width of each segment in the device. This must be a multiple of the manufacturing grid.

Number of segments
Enter a default integer number of segments.

Segment type
Choose either Series or Parallel.

Add an N-Well shield under the resistor
This option will create two columns of contacts.
Add an extra column of contacts to the heads of the resistor

- Adds an extra column of contacts to the heads of the resistor.

Maintain uniform current flow direction through the resistor segments

- Applies to series resistors only.

Add a dummy segment to each end of the resistor

- Adds dummy resistors for process matching purposes.
Resistor Page 2

Rotate the resistor by 90 degrees
Rotates the entire resistor by 90 degrees.

Width of rail to join resistors
The width of the rails used to join the resistors together. This must be an integer multiple of the manufacturing grid. These are the metal wires above the resistor array.

Width of metal at resistor head
The width of the metal added to the head of a resistor when the unit resistor is made up from an odd number of segments and the current flow direction is non-uniform.

Number of rows
The number of rows in the resistor array. This must be a positive integer number.

Generate a Resistor or Matched Resistor Array
Once you have completed the resistor definition you can generate an instance by dragging the cell name from the Design navigator into the layout to open the dialog shown below. You can also use this dialog
to generate a matched resistor array.

**Ports**

A list containing at least two elements, which contains the names of the ports within the resistor array. The port names must exist within the connectivity network.
Device Generation From an SDL Netlist

HiPer DevGen can identify certain schematic configurations and will create the appropriate layout if you have drawn your schematics to conform to the HiPer DevGen identification criteria. (See “Schematic Characteristics that Define a HiPer DevGen Device” on page 439.)

First you define HiPer DevGen T-Cells in L-Edit. Next you design a schematic in S-Edit with the necessary parameters attached to the symbols, and export it as a SPICE netlist. When you open the netlist using the L-Edit schematic driven layout (SDL) navigator, SDL passes those parameters to the HiPer DevGen T-Cells to automatically generate parameterized layout.

The following example illustrates how to set parameters in an S-Edit schematic to pass to HiPer DevGen through SDL.

Network

A hierarchical list specifying the connectivity within the resistor array. This must contain at least one element. Each element of the list is structured as follows:

network[0] - this specifies the chain type and can either be series (route the chain in series) or parallel (route the chain in parallel).

network[1] - this is the number of unit resistors in that gain. This must be a positive integer number.

network[2] - this must be a list containing exactly two elements, which can be a string of any type.

The first element contains the name of the net in which the Rp terminal of the first resistor in the chain (or resistors in the case of a parallel connection) will be connected.

The second element contains the name of the net in which the Rm terminal of the last resistor in the chain (or resistors in the case of a parallel connection) will be connected.
In this opamp example, the instance M1 is part of a differential pair and has the related HiPer DevGen properties assigned.

Similarly, instance M9 is part of a current mirror that also includes M10 and M11, and has current mirror properties assigned. When you enter properties for a given device type, HiPer DevGen can determine which devices are part of a differential pair or current mirror. Note that only one symbol in a device type needs to have properties assigned. S-Edit will combine the properties for those symbols into a single line when it sends it to the netlist.

When you export the cell to a SPICE netlist, the Export control property must be set to SDL. Tanner EDA has special properties that export different parameters depending on whether you are using the
netlist for simulation or SDL. This is specific to our libraries. If “SDL” is not in the drop down list, simply type it in.

![Export SPICE dialog box]

<table>
<thead>
<tr>
<th>General</th>
<th>Hierarchy Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>To file</td>
<td></td>
</tr>
<tr>
<td>Y:\My Documents\My_HipEditGen.spc</td>
<td></td>
</tr>
<tr>
<td>Confirm overwrite</td>
<td></td>
</tr>
</tbody>
</table>

**Export source**
- **Design:** OpAmp
- **Cell:** OpAmp_TestBench
- **View:** schematic Transconductance Interface: Trans

**Export as**
- **Top-level cell**
- **Subcircuit definition**

**Export mode**
- **Hierarchical**
- **Flat**

**Options**
- **Exclude model**
- **Exclude and**
- **Exclude simulation commands**
- **Create separate file for each corner**
- **Wrap lines to** 80 characters

**Export control property**
- **Property name:** SPICE
You can click on the netlist filename hyperlink in the S-Edit command window to open the exported SPICE netlist.

Your L-Edit .tbd file must contain the necessary tech setup and cell library (MOSFETS, contact cells, etc.) or a version of the correct type of HiPer DevGen T-Cell cell defined in your file.
Import the netlist into L-Edit using the SDL navigator. In the Layout tab of Import Netlist, you must enable Log all changes, Add instances for missing subcircuits, Add instances of T-Cells for missing devices using the Model name, and Update parameters of T-Cells.

Additional X-ref libraries must also be set to the appropriate technology setup file for your HiPer DevGen process if the T-Cell are in a different library file than the one you currently have open.

In the HiPer DevGen tab of Import Netlist, you must list the node names you want to use for power and ground. Make sure to enable Recognize differential pairs and Recognize current mirrors.
Enter the model names and the corresponding HiPer DevGen T-Cell names for both device types. Note that this option only supports one T-Cell per device model. In other words, you cannot have NMOS25 be mapped to two different T-Cells.

L-Edit will create and open a log describing the import outcome. Note that M5 and M6 are correctly identified as comprising a PMOS current mirror, with a warning issued because the unit multiplicity,
with a value of 1 is odd, so matching might be incorrect. M7 and M8 are similarly identified, and M9, M10 and M11 are found at the bottom for the other current mirror along with the two differential pairs.

L-Edit automatically creates the device layout. Note that instance placement directly corresponds to the relative symbol position in the schematic.
The SDL navigator shows all the nets that were part of the netlist from the schematic. You can select nets and press the flylines button to show the connectivity when you route the cells.

Schematic Characteristics that Define a HiPer DevGen Device

HiPer DevGen recognizes common transistor structures such as current mirrors and differential pairs, and will combine individual transistors in an S-Edit schematic to construct these structures. If the circuit is drawn to satisfy the required conditions, a designer can use existing schematics without modification.

S-Edit parameters entered for the schematics are saved in the L-Edit file as cell properties. The parameters of the merged structure are simply the union of the parameters of the individual devices in S-Edit. You can control advanced matching and other attributes of the merged structure by setting a single parameter on one of the constituent transistors.

Typically the length (L), width (W) and number of finger (NF) parameters have to match between all components devices, and other parameters, for example (WPE) are merged and therefore need only be present on one of the component devices to control the setting for the differential pair or current mirror as a whole.
During HiPer DevGen SPICE import, L-Edit checks the schematic for certain characteristics as described below.

**Schematic Criteria for a Current Mirror**

Each of these criteria must be true for HiPer DevGen to recognize a schematic circuit as a current mirror:

- Is there a diode-connected MOS, whose source terminal is connected to a supply rail? If so, collect all devices that share a gate connection with the diode.
- Are they the same type as the diode?
- Do they have the same gate length as the diode?
- Is their source terminal connected to the same supply rail as the diode?
- Is their number of fingers the same as the diode?
- Is their width an integer multiple of the diode width?
- Is the number of fingers the same?

**Schematic Criteria for a Differential Pair**

Identifying a differential pair is more difficult than identifying a current mirror, as it requires finding and comparing two devices, not just one device with special properties.

First, we collapse parallel MOSFETS. Then, each of these criteria must be true:

- Do exactly two devices share a source terminal, that is not a supply rail?
- Are both devices of the same type (model)?
- Do they have the same number of fingers and multiplicity?
- Do they have identical gate lengths and identical gate widths?
- Are their gate terminals connected to different nets?
- Are their drain terminals connected to different nets?

**Schematic Criteria for a Resistor Array**

For HiPer DevGen to identify a resistor array, the resistors must be the same size, they must be of the same model (type) and have the same values. However, they do not need to be connected, and in fact any resistor with the same model and values will be considered a DevGen block.

Each of these criteria must be true:

- Are the devices the same type size?
- Are the devices of the same type (model)?
- Do they have the same gate length and width?
HiPer DevGen Process Setup

Process setup for each HiPer DevGen device class includes critical factors such as layer assignments, design rules and electrical parameters. Once setup is complete, HiPer DevGen device parameters are saved in the current .tdb file.

There are twenty-one setup pages in the wizard, grouped into two categories, layer setup and design rule setup.

It is important that you define design rules to be consistent with the rules saved in the current layout database file or those specified by your foundry. For example, if the capacitor top plate layer is mapped to Poly, then the minimum width of the top plate should be greater than or equal to the minimum width of Poly. Otherwise, DevGen devices may generate DRC errors or fail to fabricate properly.

The HiPer DevGen setup process establishes rules for all devices (i.e. NMOS, PMOS, capacitors and resistors). Once you have defined the setup, all devices will copy this process information.

HiPer DevGen creates keepouts for the transistors and a Metal1 keepout the resistor rnp and other layers as follows.

- Metal1_keepout - the MBB of all metal1 objects
- Metal2_keepout - a copy of all Metal 2 objects
- Metal3_keepout - a copy of all Metal 3 object.
- Metal4_keepout - a copy of all Metal 4 objects
- Metal5_keepout - a copy of all Metal 5 objects
- Metal6_keepout - a copy of all Metal 6 objects

The setup values you enter are only defaults. Should you want to modify the process design rule for a particular device, you can use Cell > DevGen T-Cell > Edit DevGen Info > Setup Process to do so, as this process information is specific to that device alone.

Cell > DevGen T-Cell > Setup HiPer DevGen Process Information launches the wizard that guides you through the process setup.
Additional setup information includes the names of the default device classes for capacitor, inductor, resistor, and N- and P-Channel MOSFET devices and the layer on which ports are placed and the size and alignment of text.

Layer Assignments

Pages two through six are layer assignments that map DevGen layer names to existing layers in your L-Edit database file as well as specifying via layers, routing layers, diffusion layers, etc. To assign a target layer, select from the drop-down list.
Design Rule Assignments

Pages seven through twenty-five are design rule assignments.

![Dev-Gen Setup Wizard](image)

<table>
<thead>
<tr>
<th>Rule Description</th>
<th>Value (Microns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Metal 4 Width</td>
<td>0.55</td>
</tr>
<tr>
<td>Minimum Metal 4 to Metal 4 Spacing</td>
<td>0.500</td>
</tr>
<tr>
<td>Wide Metal 4 Width</td>
<td>1.200</td>
</tr>
<tr>
<td>Minimum Metal 4 to Wide Metal 4 Spacing</td>
<td>1.000</td>
</tr>
</tbody>
</table>
12 Derived Layers

There are two types of layers in L-Edit, drawn layers and derived layers, also known as generated layers. A drawn layer is a layer that the designer will draw on.

A derived layer is generated by L-Edit based on operations and selections made to existing layers. While you can draw on a derived layer, all polygons on a derived layer will be deleted prior to generation if that layer is used to derive others.

You can change a derived layer to a drawn layer simply by setting it to Drawn type in Setup Layers—Derivation. L-Edit saves the derivation in case the layer is reclassified as Derived.

Layer Derivation Steps

To first step is to create a derived layer using Setup > Layers—Derivation. To do this you add a name for the derived layer to the layer list, and then define its characteristics. The target layer—the layer on which the operation results will be drawn—is always the layer highlighted in the Layers list.

Any layer in the layer list that is above the one to be derived, including other derived layers, may be used in the definition of a derived layer. The layer that you want to generate must always be inserted in the layer list after the names of the layers that are used to create it.

The next step is to generate the derived layers using Tools > Generate Layers. You choose the command set (L-Edit Standard-Derived layers or layer definitions from Calibre, Assura or Dracula). Objects on derived layers are automatically created during the generate layers operation (see “Generating Derived Layers” on page 458.)
Defining Derived Layers

You define generated layers in the Derivation tab of Setup Layers dialog. To open this dialog, choose Setup > Layers, or double-click anywhere on the layers palette.

![Setup Layers dialog]

Derived layers can be created using the following operations:

- **Boolean** — Applies the logical AND, OR, and NOT operators to a combination of source layers.
- **Select** — Selects polygons based on their area relationships between source layers.
- **Area** — Selects polygons that are either equal to a specified area or within a specified area range.
- **Density** — Selects polygons based on the relative density of two source layers.

Each derivation type includes a Boolean NOT case, which provides the complimentary output of a given operation.
To define derived layers using Boolean operators, click on the Derivation tab in Setup Layers and select Boolean from the Derivation Type drop-down menu.

Use this dialog to pick the name of the derived layer, specify up to three source layers, and compose the Boolean operations performed to create the new layer. Note that operations are always evaluated from top to bottom, and the derived layer must be positioned in the Layers list below each of its source layers.

**Type**

Choose the Boolean derivation type from the drop-down list.

**Show**

Opens the dialog Full Derivation, which shows the derivation for the selected layer and all of its source layers in terms of drawn (mask) layers.

**Source layers (Display Units)**

Existing layers from which the new layer will be created (derived). Select each source layer (Layer 1, Layer 2, Layer 3) from the drop-down list. Only layers listed above the target layer are available.

Two operations can be applied to each source layer:

- **NOT**—when this box is checked, the complement of the source layer is used.

- **Grow**—enter a positive or negative integer for the amount, in Display units, by which objects on the source layer are grown or shrunk on the derived layer. Objects grow or shrink uniformly by the given quantity.
Boolean Operation Outcomes

Three elementary Boolean operations can be applied to previously defined layers: **AND**, **OR**, and **NOT**. L-Edit also performs the “Grow” (page 448) operation, used to oversize or undersize (shrink) objects. These operations can be used individually or combined to produce more complex formulas.

**AND**

The **AND** operation (abbreviated &) creates objects on a derived layer from the intersection of objects on two other layers.

**OR**

The **OR** operation (abbreviated |) creates objects on a derived layer from the union of objects on two other layers.

**NOT**

The **NOT** operation creates objects on a derived layer based on the absence, or inverse, of objects on another layer. Mathematically, the derived layer should extend throughout the layout area wherever the
original layer does not exist. Because the layout area is not explicitly defined, L-Edit applies the NOT operation within the minimum bounding box that encompasses all existing objects.

Grow

The Grow operation creates objects on a derived layer by increasing the size of each object on the original layer. Specifically, a Grow operation displaces each edge by the specified number distance (in display units).

A negative Grow parameter yields a shrink operation, creating objects on a derived layer by displacing the edge of each object on the original layer inward by the specified distance (in display units).

If any dimension of an object is less than or equal to twice the shrink amount, a new object will not be created on the derived layer.
Boolean Order of Operations

Boolean operations are performed on the source layers in the following order:

13. Grow (individually)
14. NOT (individually)
15. AND/OR (first to last)

AND has higher precedence than OR. For example, \( a \text{ AND } b \text{ OR } c \) is read as \((a \text{ AND } b) \text{ OR } c\), and \( c \text{ OR } a \text{ AND } b \) is read as \(c \text{ OR } (a \text{ AND } b)\).

In the following example, the source layers consist of layers \(a\), \(b\), and \(c\), with the following operations: NOT and Grow 2 on \(b\); AND between \(a\) and \(b\); OR between \(b\) and \(c\). The total derivation can be expressed as \((a \text{ AND } (\text{NOT}(\text{Grow 2}(b)))) \text{ OR } c\).

In this case, the operations are performed as follows:

1. Grow 2 on \(b\)
2. NOT on the result of step 1
3. AND between \(a\) and the result of step 2
4. OR between the result of step 3 and \(c\).
Select Layer Derivations

Select operations allow you to define a relationship that selects a group of polygons from a layer and creates a new layer with the results. These operations allow you to create rules that cannot be made with logical operations, such as spacing checks and sizing checks.

Choose the Select derivation type from the drop-down list.

Opens the dialog Full Derivation, which shows the derivation for the selected layer and all of its source layers in terms of drawn (mask) layers.

Existing layers from which the new layer will be created (derived). Select each source layer from the drop-down list. Only layers listed prior to the target layer are available.

When this box is checked, the NOT of the relation is applied.
Select Operation Relationships

The select relationships are described and illustrated below.

Inside

The inside operation selects layer 1 polygons that are completely contained in layer 2 polygons, as shown below (where layer 1 is red).
Not Inside

The **not inside** operation selects layer 1 polygons that are NOT completely contained in layer 2 polygons (equivalent to **outside** or **cut**), as shown below (where layer 1 is red).

Outside

The **outside** operation selects layer 1 polygons that are completely outside of layer 2 polygons, as shown below (where layer 1 is red).
Not Inside

The **not outside** operation selects layer 1 polygons that are not completely outside layer 2 polygons (equivalent to **inside** or **cut**), as shown below (where layer 1 is red).

![Not Inside Diagram](image)

Hole

The **hole** operation selects layer 1 polygons that have their entire outside surface exactly touching the outside surface of a layer 2 polygon, as shown below (where layer 1 is red).

![Hole Diagram](image)

The **not hole** operation selects any layer 1 polygons that do not exactly fill a hole in a layer 2 polygon.
Cut

The **cut** operation selects layer 1 polygons that intersect but do not just touch layer 2 polygons, so that they have areas that are both inside and outside of layer 2 polygons, as shown below (where layer 1 is red).

![Cut Diagram](image1)

Not Cut

The **not cut** operation selects layer 1 polygons that are either completely inside of or completely outside of layer 2 polygons, as shown below (where layer 1 is red).

![Not Cut Diagram](image2)
Touch

The **touch** operation selects layer 1 polygons that touch layer 2 polygons from the outside and do not also cut, as shown below (where layer 1 is red).

Not Touch

The **not touch** operation selects layer 1 polygons that do not touch layer 2 polygons, as shown below (where layer 1 is red).

Enclose

The **enclose** operation selects layer 1 polygons that completely enclose layer 2 polygons. This includes layer 1 polygons that are inside and touching layer 2 polygons. The **not enclose** operation selects layer 1 polygons that do not completely enclose layer 2 polygons.

Overlap

The **overlap** operation selects layer 1 polygons that touch, cut, enclose or are inside of layer 2 polygons.

The **not overlap** operation selects layer 1 polygons that are entirely outside of layer 2 polygons.
Chapter 12: Derived Layers

Defining Derived Layers

Vertex

The **vertex** operation selects layer 1 polygons with more than or equal to a specified minimum number and fewer than or equal to a specified maximum number of vertices. The **not vertex** operation selects layer 1 polygons with fewer than the minimum number or more than the maximum number of vertices.

Area Layer Derivations

The **Area** option in the Derivation tab checks polygons on the selected layer to determine if their areas are equal to a specified area or are within a specified area range.

![Setup Layers](image)

**Type**  
Choose the **area** derivation type from the drop-down list.

**Show**  
Opens the dialog **Full Derivation**, which shows the derivation for the selected layer and all of its source layers in terms of drawn (mask) layers.

**Source Layers**  
Existing layers from which the new layer will be created (derived). Select each source layer from the drop-down list. Only layers listed prior to the target layer are available.

**NOT**  
When this box is checked, the NOT of the relation is used, so polygons with area outside the specified range are flagged.

**Square Display units** or **Square Lambda**  
Select one of these radio buttons to set the units for area calculation. As with the rule distance in **Setup Design Rules**, (see “Specifying DRC Standard Design Rules” on page 542) areas do not get rescaled when **Square Display Units** is selected, but do get rescaled when **Square Lambda** are used.
Density Layer Derivations

Certain design rules require testing of the percentage of area covered by a certain layer. The density operation selects polygons based on the percentage of area that one layer covers in an area defined by another layer. The selected objects can then be used in a density DRC rule.

The density operation derives boundary polygons within which the density of layout on the Density of layer is between a minimum \( n_1 \) and maximum \( n_2 \) percentage. The boundary may be the cell extent, or may be polygons on a specified layer. The density of the input layer is computed separately within each boundary polygon by performing a Boolean AND operation of the input density layer with each polygon on the boundary layer. Valid range values are between 0 and 100 percent, where \( n_1 \) must be less than or equal to \( n_2 \).

The not density operation selects boundary polygons within which the density of layout on the input Density of layer is less than the minimum percentage or greater than the maximum percentage.

L-Edit calculates the area density of the specified density layer that exists inside each polygon of the boundary layer, and outputs the boundary layer polygon if the density is within the specified range.

Specifically, for each polygon \( P \) in layer 1, L-Edit calculates:

\[
density = \frac{\text{area of } (P \text{ AND layer 2})}{\text{area of } P} \times 100\%
\]

- then outputs \( P \) to the derived layer if the density is in the specified range (or not in the given range if NOT is turned on).
The options used in designating the density layer are:

**Derivation Type**
Choose **Density** from the drop-down list.

**Show**
Opens the dialog **Full Derivation**, which shows the derivation for the selected layer and all of its source layers in terms of drawn (mask) layers.

**NOT**
When this box is checked, the NOT of the relation is applied.

**Density of layer**
The input layer from which the density is calculated.

**Inside polygons of layer**
The boundary polygons within which the density is measured. The boundary can be a layer or the cell boundary.

**n1, n2**
Enter a minimum (n1) and maximum (n2) value (exclusive) to define the density range.

---

**Setting Up Command File Derived Layers**

Layers can also be defined in command files, using Dracula, Assura or Calibre command file format, using any of the layer derivation commands. See “HiPer Verify: Calibre Command Files” on page 557 or “HiPer Verify: Dracula Command Files” on page 723 for details.

---

**Generating Derived Layers**

After derived layers have been defined, they can be generated in the active cell using **Tools > Generate Layers**.

**Choose a Command Set**

- Invoke **Tools > Generate Layers**
- If you want to generate layers defined in the **Setup Layers** dialog, select **Standard-Derived Layers**. If you want to generate layers defined in a Dracula, Assura or Calibre command file, press the **Add Command File to List** button ( ), then press the Browse button ( ) to browse to and select a command file from disk.
- You can setup and save multiple command files, but only one at a time can be used. The derived layers defined in L-Edit or in a command file are listed in the lower portion of the dialog, for the highlighted command set. Command files containing both DRC statements and derived layers can be included in the list.
- Highlight the command file you wish to run.

**Run the Derivation**

- Check the checkboxes corresponding to the layers you wish to generate.
- Press the **Run** button to generate the layers you have selected, or **Accept** to save your settings without running the derivation.
Command Sets

Enter the command files you wish to run, or select **Standard-Derived Layers** to generate layers derived in the Setup Layers dialog. Use the **Add command file to list** and **Delete command file from list** buttons to add and remove files from the list.

Select the command set containing the layer definitions you wish to generate.

Layers to generate

Place a checkmark next to each layer you wish to generate. You can mark layers individually by clicking in the checkbox next to the layer name. To mark a group of adjacent layers, click and drag the mouse from the first to the last layer in the group. Use **Mark All** and **Unmark All** to add or remove checkmarks next to all layers in the list.

Only derived layers for which derivation is enabled may be generated. Locked layers and derived layers that have the **Enable Derivation** option off are not listed in this dialog. Ports on derived layers are not deleted.
When you execute the **Generate Layers** command, L-Edit automatically deletes existing objects on generated layers before regenerating those layers. Only those layers selected for generation will be cleared.

When layers are generated from command files, the following behavior will occur:

- If a layer does not exist in L-Edit for the layer being generated, then a new layer will be created at the end of the layer list with the name of the layer being generated, and the geometry is put on this layer. The layer is created as Derived, with Type External.

- If a Derived layer of type External already exists in L-Edit with the same name as a layer being generated, then this layer is reused.

- If a Drawn layer exists in L-Edit with the same name as a layer being generated, then a new layer will be created at the end of the layer list with the name of the layer being generated plus ".1" appended, and the geometry is put on this layer. The ".1" is incremented as required to avoid name collisions. The layer is created as Derived, with Type External.

- If a Derived layer of type other than External exists in L-Edit with the same name as a layer being generated, then it is treated the same as a Drawn Layer.

**Automatic Layer Generation with DRC and Extract**

L-Edit automatically generates objects on derived layers before DRC or Extract runs and clears those objects afterwards. It only deletes objects generated during that DRC or Extract run, however—previously generated objects remain.
Chapter 12: Derived Layers

Showing and Hiding Generated Layers

Use the menu commands View > Layers > Show Generated and View > Layers > Hide Generated to show and hide derived layers. (See “Hiding Layers” on page 192.) If the current layer is a derived layer, it will remain visible when derived layers are hidden.

You can also show or hide derived layers using the context-sensitive menus on the layer palettes, just as you would any other layer. With the pointer over any non-derived layer icon, click the right mouse button and choose Hide Generated. All derived layers will be hidden and their icon or name will be shaded on the layer palettes.

To hide all derived layers except the selected layer, position the pointer over the desired derived layer icon and choose Hide Generated in the context-sensitive menu. Choose Show Generated to display all generated layers in all cells for the active file.

Locking Generated Layers

You can lock derived layers from the Layer palette just as you would any other layer. Select “generated” from the pull-down list then right-click on any layer name to open the menu. Pick “Lock” and then “Lock All” or lock layers individually by clicking in the Lock ( ) column.

Removing Generated Layers

Tools > Clear Generated Layers removes objects from all derived layers in the active cell, the active file, or in all open files. All objects on derived layers are deleted, regardless of how they were created. Ports may be optionally kept. This command cannot be undone.

This Cell (cell name)
Removes derived layers in the active cell.

This File (file name)
Removes derived layers in the active file.

All Files
Removes derived layers in all open files.

Do not delete ports
Prevents L-Edit from removing ports on the derived layers in the specified file or cell. Use this option when your design uses a derived layer as its extract recognition layer and you want to retain the ports on this layer.
13 Schematic-Driven Layout (SDL)

L-Edit Schematic Driven Place and Route (SDL) provides assisted manual and automatic place and route tools for mixed and standard cell blocks. Specifically, the following features are provided:

- Manual placement of mixed size instances, assisted by flyline display of connectivity.
- Manual routing of mixed or standard cell blocks, assisted by dynamic flyline display.
- Automatic placement of standard cell blocks.
- Automatic routing of mixed size or standard cell blocks. An entire block may be routed at once, or selected nets may be incrementally routed.
- Clock routing and Power/Ground routing of standard cell blocks.

The router conforms to the following fundamental rules:

- Traces and ports are routed center-to-center on routing grid points.
- Routing wires are orthogonal.
- The routing layer that connects to a port is determined by the layer on which the port is drawn.
- Wherever possible, horizontal routing is performed on odd-numbered routing layers and vertical routing on even-numbered layers.

SDL reads a netlist for instance and connectivity information, then places instances into a cell, creates flylines, and can perform automatic routing that generates orthogonal routing wires and vias between devices and I/O ports.

SDL also reads LEF files for routing layers, vias definitions and obstructions, then attaches cell and view names in the design with those in a LEF file to create composite cells having the proper instance and port geometry. The automatic placer inserts clock buffers and can be weighted for parameters like density, processing time or compactness.

The SDL Navigator provides quick access to assisted- and automatic place and route commands and lets you manage nets and pins during the related processes.

Assisted Place and Route

L-Edit’s assisted routing tool reads a netlist for instance and connectivity information, then generates the instances into a cell, creates flylines, and performs automatic routing that generates orthogonal routing wires and vias between devices and I/O ports.

The router is a gridded area router – the center and end-points of routing wires are always placed on-grid. You can set the router to automatically calculate the routing grid, or you can set it manually to adhere to a wire-to-wire, wire-to-via or via-to-via spacing domain.

Assisted routing supports an unlimited number of layers, with user-specified width and spacing rules for each layer and the option to specify obstructions on a per-layer basis. You can choose to route single nodes, groups, or the entire design on any given pass. When you use assisted routing to generate layout
corresponding to subcircuits and devices, L-Edit will dynamically update the flylines as you move instances.

If you have nets that are critical to a design you can auto-route them first to benefit from the maximum availability of routing resources, or you can route them manually to satisfy specific performance requirements and then let the router automatically route the rest or the design.

In addition to incremental routing within the layout, assisted routing has an engineering-change-order (ECO) capability so you can input netlist changes without disturbing existing placement and routing.

Below is a very simple example of SDL-generated flylines prior to routing.

### Using the Assisted Router

**Configure for Optimal Routing**

**Layers** – Set the routing layers based on your standard cell library. See “Define the Routing Layers” on page 479.

**Grid** – Determine whether you should specify a routing grid spacing or have L-Edit calculate one. See **Set Grid Spacing** (page 483).

**Port Placement** – Place ports on cell boundaries and, if necessary, on grid. See “Locating Ports” on page 487.

**Obstructions** – Add obstructions or holes in obstructions. See “Obstruction Regions” on page 490.

**Netlist** – Prior to running SDL, check that each device listed in your netlist is a cell with pre-defined layout or a T-Cell. The device name in the netlist must match the cell name in your database file for that device to be instantiated. See “Netlist Format for Assisted Routing” on page 492.
Import a Netlist

“The SDL Navigator” (page 464) is a dockable window with a toolbar for all the SDL commands. It has a Load Netlist icon that opens the “Import Netlist” dialog, where you specify which netlist to load and which layout elements to add or update if they are not included in the netlist—see “Importing a Netlist” on page 471.

Cells and ports that do not exist or do not match by name can be optionally generated when you import the netlist. See “Automatically Generating Layout Elements” on page 474.

Position Layout

Once netlist information is imported, all devices are instanced in the layout. The SDL Navigator will list all nets and their routing status.

Route Critical Nets, Auto-Route Remaining Nets

As a guide for manual placement, each net has flylines of a different color to illustrate connections and congestion. Using the flylines as a guide to minimize path length, you can manually route critical nets first. If you tag them with the netname of the active net, the router will ignore them. Use “Route All” to automatically route and label nets according to your established routing layers and settings. The router will ignore nets that have been checked off in the SDL checklist. You can rip-up and re-route by net if needed (see “Using the SDL Navigator Checklist” on page 494.)

Check for Shorted Nets

L-Edit can check routed layout against the netlist from the SDL Navigator to find shorted, open, unknown or missing nets. See “Standard Cell Placement and Routing” on page 495.

Enter Engineering Change Orders (ECOs)

If you need to import an updated netlist, SDL provides a comparison function that displays added or deleted pins and nodes that have changed. See “Updating from an Engineering Change Order (ECO)” on page 473.

The SDL Navigator

The SDL Navigator is a dockable toolbar for schematic-driven layout, automatic placement and automatic routing commands. It also displays information about routing status in a checklist format. The checkboxes can be used to mark nets to indicate that they are completed and therefore excluded from further routing operations or to automatically display status.

Launch the SDL Navigator by right-clicking in the main toolbar area and selecting Docking Views > SDL Navigator, using the menu option View > Docking Views > SDL Navigator, or by checking the option Tools > SDL Navigator > Show SDL Navigator.
Chapter 13: Schematic-Driven Layout (SDL)

The SDL Navigator Toolbar

Netlist View
- **By Net**—Displays a list of nodes present in the current cell. Expanding a node shows all the pins connected to that node. The format of each pin is “instance name”/“port name”.
- **By Instance**—Displays a list of instance names present in the current cell. Expanding an instance name shows all the nodes connected to that instance. The format of each node is “port name”="node name”.
- **By Unrouted Segment**—Displays a list of nodes in the current cell. Expanding a node shows the coordinates of the segments that could not be automatically routed to each other.

Load Netlist
Opens the **Import Netlist** dialog. (See “Importing a Netlist” on page 471.)

The SLD Navigator Command Menu

- **Toggle Markers**
- **Remove All Markers**
- **Assisted Routing**

Filter field
- A red X indicates that all or a portion of the net could not be routed.
- A green check indicates that the entire net was routed successfully.

Netlist View displays a pull-down menu of netlist view options. (These view options are also available from the SDL Command Menu and the main menu bar.)
Chapter 13: Schematic-Driven Layout (SDL)

The SDL Navigator

Marker

Adds or removes markers from the currently highlighted items in the SDL Navigator. When you move an instance while markers are displayed, the marker will update based upon the new placement of the instance.

- When viewing **By net**, if a node is selected in the tree view, then markers are toggled at all pins within the node. If a single pin is selected in the tree view, then a single marker is toggled at that pin.
- When viewing **By Instance**, if an instance is selected in the tree view, then markers are toggled at all nodes connected to that instance. If a single node is selected in the tree view, then a single marker is toggled at the node connection point to the instance.
- When viewing **By unrouted segments**, if a node is selected in the tree view, then a set of markers will be toggled for each unrouted segment of that node. If segment coordinates are selected in the tree view, then a single marker is toggled at that segment.

Flyline

When a node is selected, click to display a set of flylines that connect all pins connected to the node.

When you move an instance while flylines are displayed, the flyline will update based upon the new placement of the instance.

- If a **pin** is selected, the flylines for its parent node are shown.
- If an **instance** is selected, the flylines for all nodes connected to that instance are shown.
- If **unrouted segment** coordinates are selected, the flyline for the segment is shown.

L-Edit flylines use a *minimum spanning tree* showing the shortest connection lengths necessary to connect all the desired pins while minimizing the orthogonal distance between connected pins.

Highlight Layout

Toggles highlighting of routing layout (wires) for the items highlighted in the SDL Navigator.

- If a **pin** is highlighted, wires for its parent node are shown.
- If an **instance** is highlighted, wires for all nodes connected to that instance are shown.

Zoom

Zooms to the highlighted entry.

- If the entry is a **pin**, zooms to the pin.
- If the entry is a **node**, zooms to the MBB of all the pins on the node.
- If the entry is an **instance**, zooms to the instance.
- If the entry is an **unrouted segment**, zooms to the MBB of the coordinates of the segment.

Note that you can also zoom to markers and flylines using the “**w**” hotkey (as long as the selection list is empty).

Route All

Routes all unchecked nodes using the SDL automatic router.
Command Menu

Opens a pull-down menu of place and route commands. See “The SLD Navigator Command Menu” on page 469.

Toggle Markers

Toggles visibility of all markers and flylines in the display.

Remove All Markers

Clears all markers and flylines from the display. Note that this operation cannot be undone.

Assisted Routing

Turns on/off Assisted Routing features.

(Filter field)

Typing in the filter field searches the display list and highlights the first item that contains the characters entered. The item can be in any part of the tree, and the match is case-insensitive. The search begins with the item currently selected.

Right-Click Menu Commands

Right-click on any item in the SDL Navigator to open a context-sensitive menu. The controls will vary depending on whether a node, pin or instance is selected.

Active Net

Selects or deselects the currently highlighted items in the SDL Navigator as the active net. Any objects or instances that are placed in the layout while a net is active will be tagged with that net name. See “Setting the Active Net” on page 479.

Marker

Adds or removes markers from the currently highlighted items in the SDL Navigator.

Flyline

Adds or removes flylines from the currently highlighted items in the SDL Navigator.
Highlight Layout
Highlights the wires, vias or instances connected to the currently selected items in the SDL Navigator.

Route
Routes the selected nodes using the SDL automatic router. See “Using the SDL Navigator Checklist” on page 494.

Route only on
Use this option to select one routing layer for single-layer routing.

Ripup Net
Removes all objects tagged with the selected node name.

Ripup Instance
Removes the selected instances if they were placed automatically. (An instance that you have placed manually will not be affected by this command.)

Select Net
Selects the tagged geometry in the layout associated with the highlighted items in the SDL Navigator.

• If a node, pin, or unrouted segment is highlighted, any tagged geometry for the associated nets will be selected.
• If an instance is highlighted, all tagged geometry for the nets connected to the instance will be selected.

Select Instance
Selects the layout instances associated with the highlighted item in the SDL Navigator.

• If a node is highlighted, instances the node connects to will be selected in layout.
• If a pin is highlighted, the instance the pin is connected to will be selected in layout.
• If an instance is highlighted, that instance will be selected in layout.
• If an unrouted segment is highlighted, the instances that segment connects to will be selected in layout.

Jump to schematic
When a net or instance is selected in the SDL Navigator, use this command to highlight that net or instance in S-Edit.

Mark Complete/Unmark
Complete
Toggles the checkmark for the selected node, pin or instance to indicate completion of routing.

Power
Toggles indication of the selected node names as the power node (shown with “(power)”after the node name.

Delete
Removes the selected node, pin or instance from the SDL Navigator window.

Note: This operation cannot be undone.

Highlight All
For any of the display modes, highlights all items listed in the SDL Navigator.

Collapse Tree
Closes all open sub-trees in the navigator window.
The SLD Navigator Command Menu

The Command Menu button in the SDL Navigator ( ) provides the most comprehensive access to SDL router commands.

Add Selection Flyline
Adds flylines for all nodes in the active layout selection. Select CTRL+A (select all) in the layout and use this command to display all flylines for the cell.

Tag Selections with active net
Designates any objects selected in the layout as part of the active net. See “Setting the Active Net” on page 479.

Highlight Selected Net
Highlights the net(s) corresponding to the currently selected objects in the layout. Objects must be tagged with net identification.

Pick Closest Pin
When you invoke this command, L-Edit prompts you to click the mouse in the layout window. The pin closest to where you click will be highlighted in the netlist tree view.

This command is particularly useful to identify nodes, and then, with the assistance of the flyline tool, show interconnections.

Ripup All Unmarked Nets
Removes all objects that are tagged with a routing node name. Only removes objects on nets that are not checked. See “The SDL Navigator” on page 464.

Ripup Placed Instances
Removes instances placed by schematic driven layout. Locked instances are not removed.
Chapter 13: Schematic-Driven Layout (SDL)  The SDL Navigator

Tools > SDL Navigator Commands

You can also use **Tools > SDL Navigator** from the menu bar to access the commands shown below.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove ECO status</td>
<td>Removes the ECO icon indicators and updates the SDL Navigator using the latest ECO netlist. See “Updating from an Engineering Change Order (ECO)” on page 473.</td>
</tr>
<tr>
<td>Export Wire List...</td>
<td>Exports the SDL Navigator node list in a simple text netlist format. See “Alternative Netlist Format” on page 493.</td>
</tr>
<tr>
<td>LEF/DEF</td>
<td>See “Import LEF Technology” on page 495.</td>
</tr>
<tr>
<td>Placer</td>
<td>See “Configure the Placer” on page 500.</td>
</tr>
<tr>
<td>Clock Router</td>
<td>See “Route Clock Nets” on page 506.</td>
</tr>
<tr>
<td>Router</td>
<td>See “Route Remaining Nets” on page 506.</td>
</tr>
<tr>
<td>Extract</td>
<td>See “Check for Shorted Nets” on page 464</td>
</tr>
<tr>
<td>By Net</td>
<td>See “Netlist View” on page 465.</td>
</tr>
<tr>
<td>By Instance</td>
<td>See “Netlist View” on page 465.</td>
</tr>
<tr>
<td>By Unrouted Segment</td>
<td>See “Netlist View” on page 465.</td>
</tr>
<tr>
<td>Release License</td>
<td>Releases the SDL license immediately. Unless this command is invoked, the license is only released when you close a file.</td>
</tr>
</tbody>
</table>

**Tools > SDL Navigator Commands**

You can also use **Tools > SDL Navigator** from the menu bar to access the commands shown below.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show SDL Navigator</td>
<td>Toggles display of the <strong>SDL Navigator</strong> on and off.</td>
</tr>
<tr>
<td>Load Netlist</td>
<td>Opens the <strong>Import Netlist</strong> dialog. (See “Importing a Netlist” on page 471.)</td>
</tr>
<tr>
<td>Add Selection Flyline</td>
<td>Adds flylines for all nodes in the active layout selection. Select <strong>CTRL+`</strong> (select all) in the layout and use this command to display all flylines for the cell.</td>
</tr>
<tr>
<td>Tag Selections with active net</td>
<td>Tags any selected objects in the layout with the active net. See “Setting the Active Net” on page 479.</td>
</tr>
<tr>
<td>Route All (same as <strong>CTRL+`</strong>)</td>
<td>Routes the selected node(s) using the SDL automatic router. See “Using the SDL Navigator Checklist” on page 494.</td>
</tr>
</tbody>
</table>
Chapter 13: Schematic-Driven Layout (SDL)

Importing a Netlist

Note that while you are using this feature, a networked license is checked out to your PC, labelled “SDL”. If needed, the license can be revoked manually from within SDL with \texttt{wlladmin}.

Each layout cell can have its own netlist with which it is associated. When the active cell changes in L-Edit, the SDL Navigator also changes to show the netlist associated with the new cell. If no netlist has been associated with a cell, the navigator shows a message to that effect.

Assisted routing may be run on multiple cells using the same top-level netlist file. If the subcircuit name in the netlist does not match a cell name identically, the top-level schematic will be used from the netlist.

After a netlist is imported, all geometry arrives in the “selected” state. L-Edit will open a log file when the import process finishes.

The first instance will be placed in the lower-left corner of the MBB of the current cell, with subsequent instances placed to match as closely as possible their positioning in the schematic. Whenever possible, all instances are kept within the MBB of the current cell.

You load a netlist using either the Tools > SDL Navigator > Load Netlist menu item or the “Load Netlist” button on the SDL Navigator toolbar (\texttt{Nav}). You can also transfer a netlist directly from S-Edit to L-Edit/SDL by pressing the Publish to SDL toolbar button in S-Edit (\texttt{L}).
Invoking the **Load Netlist** menu opens the following dialog:

### From File
Enter a netlist, use **Browse** to search the file system for the desired netlist, or press **Edit** to open the currently loaded netlist file in the L-Edit text editor.

### File format
Select a file format from the pull-down menu: CDL, SPICE, T-Spice, P-Spice, Verilog or WRL.

### Include substrate terminals of semiconductor devices in netlist
Bulk terminals of semiconductor devices (M, B, Q, J and Z) can be optionally included in the netlist. In the case of schematic-driven layout, three-terminal resistors (“bulk resistors”) are read, and mapped into subcircuits.

For example, the resistor “R1 net1 net2 1K $SUB=VCC $[B] $W=6u” would be mapped onto “XRR1 net1 net2 VCC B R=1K W=6u”.

### Insert multiple devices where M > 1
When checked, schematic-driven layout places multiple instances of a device when it reads M on the device in a netlist, instead of a single device with the M parameter set.

### ECO Processing
- **Replace current netlist** - does just that.
- **Compute and display differences from current netlist** highlights node and pin changes. See “Updating from an Engineering Change Order (ECO)” on page 473.

### Clear Netlist
Removes the association between the netlist and the current cell, and clears the SDL Navigator display.

The first instance will be placed in the lower-left corner of the MBB of the current cell, with subsequent instances placed to match as closely as possible their positioning in the schematic. Whenever possible, all instances are kept within the MBB of the current cell.
Updating from an Engineering Change Order (ECO)

Managing *engineering change orders* (ECOs) is the process of receiving an updated netlist, and importing it into the SDL Navigator with the option “Compute and display differences from the current netlist” selected in the Import Netlist dialog.

This operation will display nodes that have changed with an exclamation point, pins that have been deleted with a red minus sign, and pins that have been added with a blue plus sign.

Subsequent ECO netlist imports will display changes relative to the most recently imported netlist. You can identify differences between an arbitrary pair of netlists by importing the first one with “replace current netlist” selected then importing the second as an ECO netlist. Once the ECO netlist has been imported, you can use the SDL Navigator buttons as before.

Notice in the example above that simple node renames are also displayed in this view, as shown by the renaming of “Node N_9” to “N_Rename.” Also, two pins (XXa5/Inm and XXa5/Inp) were swapped, causing both a disconnection and reconnection to nodes N_7 and N_8.
Automatically Generating Layout Elements

You have the option to create layout elements when a netlist is imported. If none of the Modify Layout options are selected, the layout is not modified in any way. After the import, added or modified elements will remain in the L-Edit selection list.

Log all changes

If this option is selected, a log file is displayed showing the operations that were performed and their success or failure.

Add instances for missing subcircuits

If this option is selected, instances are created for missing subcircuits. See “Adding Instances for Missing Subcircuits” on page 475.

Add instances of T-Cells for missing devices

If this option is selected, SPICE devices are mapped into T-Cells, and device parameters are passed to the T-Cell. See “Adding Instances of T-Cells for Missing Devices” on page 476.

- Model—Choose this naming convention when T-Cells are the only type of cell in the library that uses “model” in the instance name.

- Device_Model—Choose this naming convention when cell types other than T-Cells also use “model” in the instance name.

For example, a SPICE device “M1 a b c d MOD” will create an instance of a cell “MOD” if Model is selected, and will create an instance of a cell “M_MOD” if Device_Model is selected.
Adding Instances for Missing Subcircuits

For example, given the following SPICE input:

```
X123 In Out INV
```

an instance of cell INV would be created; the instance would be called X123. If an instance with the appropriate name (e.g., X123 in our example above) already exists in the layout, it is left untouched. If “extra” instances are found (i.e. instances that have no corresponding entry in the SPICE netlist), they are left untouched. In other words, the SDL import will not modify existing layout, but will merely add to it.

A “library” TDB file can also be specified in the Search Library field. If a cell is not found in the current file, but is found in the library file, it will be copied to the current file, and instanced upon request.

**Note:**

If the cell exists in the current file, the library file is not checked at all. As such, the assisted routing feature cannot be used to update the cells in the current design file.
Adding Instances of T-Cells for Missing Devices

When this option is selected, SPICE devices are mapped into T-Cells, and device parameters are passed to the T-Cell. If a cell of the correct name exists, but it is a regular cell and not a T-Cell, it will be used instead. It is the user’s responsibility to ensure that the necessary cells exist in the current design.

If a “library” TDB file is specified in the Search Library field, L-Edit will search there for missing T-Cells. You can use the option “Model” when T-Cells are the only type of cell in the library that includes “model” in the instance name. If other types of cells include “model” in their name you must select the Device_Model option to isolate T-Cells.

T-Cell Naming and Parameters

SDL recognizes arbitrary user-supplied parameters and passes them directly to T-Cells. Note, however, that if the SPICE netlist contains multiple parameters that differ only in case, SDL will not collapse or combine them. Or, if you have instances of mosfet T-Cells in the layout that use parameters w and l (lower case) and you receive updated T-Cells with parameters W and L (upper case), upon regeneration the T-Cell instances will have w, l, W, and L in their parameter lists.

The SPICE devices, their T-Cell mappings, pin names, and the device parameters are as follows:

<table>
<thead>
<tr>
<th>Device</th>
<th>T-Cell Name</th>
<th>Pin Names</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOSFET (M)</td>
<td>Mosfet model name</td>
<td>D, G, S, B (opt.)</td>
<td>L (in meters)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>W (in meters)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>RESISTOR (R)</td>
<td>Resistor model name, or RES if no model name present</td>
<td>PLUS, MINUS</td>
<td>R (in ohms)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>CAPACITOR (C)</td>
<td>Capacitor model name, or CAP if no model name present</td>
<td>PLUS, MINUS</td>
<td>C (in Farads)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>INDUCTOR (L)</td>
<td>Inductor model name, or INDUCTOR if no model name present</td>
<td>PLUS, MINUS</td>
<td>L (in Henrys)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>DIODE (D)</td>
<td>Diode model name</td>
<td>PLUS, MINUS</td>
<td>A (in square meters)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>JFET (J)</td>
<td>Jfet model name</td>
<td>D, G, S, B (opt.)</td>
<td>A (in square meters)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>BJT (Q)</td>
<td>Bjt model name</td>
<td>C, B, E, BULK (opt.)</td>
<td>A (in square meters)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>MESFET (Z)</td>
<td>Mesfet model name</td>
<td>D, G, S, B (opt.)</td>
<td>L (in meters)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>W (in meters)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>Transmission line (T)</td>
<td>TLINE</td>
<td>AP, AM, BP, BM</td>
<td>Z0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NL</td>
</tr>
</tbody>
</table>
Layout Placement

After a netlist is imported, all geometry arrives in the “selected” state. The first instance is placed in the lower-left corner of the MBB of the current cell with subsequent instances organized as closely as possible as they are in the schematic. Whenever possible all instances are kept within the MBB of the current cell.

The SDL import will then create instances, devices and ports within this box, if possible. The order in which instances are created is the order they are encountered in the netlist based on device groups.

Manual Assisted Routing

Pressing the Assisted Routing button in the SDL Navigator turns on behavior designed to optimize productivity when performing manual routing in the SDL environment: The following behavior is enabled in Assisted Routing mode:
[1] When the mouse is placed near a pin, a bulls eye shaped snap point is shown on the pin and drawing will snap to the center of the pin. While routing a net, only the objects in the target pins are shown as snap points. This is independent of the Object Snapping settings. (It is suggested to turn off Object Snapping when using dynamic flylines, so that additional snap points do not interfere with dynamic flyline snap points.)

[2] When the mouse is placed over a pin, the net, pin name with instance hierarchy, and layer of objects in the pin are displayed on the status bar.

[3] Drawing an object starting at a pin automatically places the object on the same layer as the pin. If the pin layer has multiple layer-purpose-pairs, the object is drawn on the correct purpose as defined in Setup > Design > Tech Layers. A via may also be placed on a pin to start a net, using the “[“ or “[” keys.

[4] Drawing an object from a pin sets the Active Net to be the net for that pin, and all geometry drawn will be on that net. If there is already an Active Net set, then only pins for the active net are displayed with snap points.

[5] A flyline is rendered from the current mouse location to the target pin. If the target pin has multiple objects, then a flyline will be drawn to each object in the pin. Flylines are shown in the color of the layer of the target pin. The target pin is automatically chosen, however a different pin can be chosen as the target by moving the mouse close to the desired pin and pressing the “*” key. If the closest pin is already the target pin, then pressing the “*” key will set the target to the next closest pin.

[6] When routing a net, alignment snap points are displayed to show horizontal or vertical alignment with each object in the target pin.

[7] If Interactive DRC is turned on, then crossing geometry in the current cell on a different net will cause the flyline to change color to indicate that a short has occurred. The color is changed to the Violation Marker Color in Setup Interactive DRC.

In addition to assisted manual routing, automatic routing of all unchecked nodes may be invoked with the Route All icon in the SDL Navigator ( ) or by selecting Tools > SDL Navigator > Route All.
Once you load a netlist, the SDL Navigator displays a checklist of all nodes and instances that were imported. If you sort the view by netlist, you can expand each node to show all the pins connected to it. If you sort the view by instance, you can expand each instance to show all the nodes it connects to.

**Note:**
Nodes that only connect to a single pin are not displayed in the SDL Navigator.

The data in the SDL Navigator is stored persistently in a library, so it is available in subsequent editing sessions.

**Setting the Active Net**

The *Active Net* function allows you to automatically add the property of belonging to a given net to geometry either before or after it is drawn. This is called “tagging” a net.

You can use the *Active Net* command to select or deselect the currently highlighted node in the SDL Navigator as the *active net*. You can also use the middle mouse button as a toggle in the SDL Navigator to activate and deactivate the *Active Net* state. When you are routing nets in layout you can also right-click a node so that geometry you draw subsequently is tagged as belonging to the active net.

The net name will be italicized to indicate that it is the active net. The context-sensitive menu for the net will have a checkmark to the left of “Active Net” as shown below.

![Active Net]()

If you specify an Active Net before geometry is drawn, any objects or instances that are placed in the layout while the net is active will be tagged with that net name.

To tag objects or instances in layout after they have been placed, set the net you want to assign them to as Active, select the objects to be tagged the layout, then use the “**Tag Selections with active net**” command.

To deactivate a net, either select another net as active, right-click the currently active net and select “Active Net” again to remove the checkmark, or select the net in the SDL Navigator using the middle-mouse button.

### Configuring the Automatic Router

**Define the Routing Layers**

Routing, Via, Obstruction, and Pin layers, for the automatic router must be defined in the **Setup Design – Tech Layers** dialog.

Routing is typically performed on alternating horizontal and vertical layers. The automatic router will only route to a port placed on a layer defined in this setup. Via arrays will be used on wide metal trace layer transitions.

Note that geometry on a hidden layer will be ignored.

Single-layer routing is also supported. Use **Router > Route all only on** from the SDL Command menu to choose from the active routing layers or the right-click menu to select just one layer.
Chapter 13: Schematic-Driven Layout (SDL)  
Configuring the Automatic Router

Enable  
An X in this box means the layer is available for the autorouter to use. You can control this setting for all layers with the Enable All and Disable All buttons.

Layer  
Name of layer on which the autorouter will perform routing.

Pin  
Name of layer on which the routing will connect to a pin. If this field is left blank, the Layer value is used. Importing LEF cells will create pin objects on this layer.

Obstruction  
Name of layer or layer-purpose pair that will behave as a keepout. You must first indicate whether you are using layers or layer-purpose pairs in this field. If you are using layer values alone, all associated purposes are considered valid. Importing LEF cells will create obstruction objects (designated by the keyword OBS in the LEF file) on this layer.

Type  
Select a layer type from the pull-down menu options: Routing, Cut, Masterslice or Other.

Width  
The minimum width allowable for a routing wire.

Spacing  
The minimum distance allowable between routing elements.

Pitch  
The minimum centerline to centerline distance between minimum width wires. This value will usually be larger than the min spacing + min width to allow for Vias. If this value is left blank, the value will be taken from the Setup Router dialog for the corresponding direction.
**Offset**

Specifies an amount by which the routing grid should be offset from the origin.

**Direction**

Indicates whether routing on a layer will be placed in the horizontal or vertical direction. If this value is left blank, the value will be taken from the Setup Router dialog for the corresponding direction.

**LEF Name**

If layer names in the LEF file match those in L-Edit, this field can and should be empty.

If they do not match, before the LEF technology file is imported you must enter the LEF layer names that correspond to those in the existing OA or TDB database.

**Default purpose for placing new objects**

If desired, select a default purpose for objects created by L-Edit. If this field is blank, L-Edit uses the following search order to find and use a layer purpose.

A “Tanner legacy purpose” is one that exists but is blank because the layer did not originally have a layer purpose.

- **Autorouted objects**: Tanner legacy purpose, drawing
- **Pin objects**: Tanner legacy purpose, pin, drawing
- **Obstruction objects**: Tanner legacy purpose, blockage, drawing

Note that L-Edit will not populate this field although it has found and is using a default purpose. If you have explicit values in this field but a matching layer-purpose pair does not exist in the LEF file, you will probably have errors significant enough to abort the cell import process.
Router Settings

Before you use the automatic router, you must specify the confines of the routing area, the grid it will use and related options. To open the Setup Router dialog, use the Command Menu ( ) button from the SDL Navigator toolbar and choose Router > Setup Router.

Routing extent polygon on layer

Sets the layer which the boundary is drawn that will confine the extent of the routing boundary. Routing will be contained within a polygon drawn on this layer.

To allow the auto-router some extra space so that it can place routing outside the MBB of the I/O ports, you can draw a box on the Routing Region layer that includes some margin space around the layout.

If no layer is specified here, or if no polygon is drawn in that layer, the default routing area is the cell MBB.

X Spacing, Y Spacing

Enter the minimum wire-to-wire spacing in the X and Y directions for the automatic and SDL routers. A zero value causes the SDL router to compute the routing grid as described in “Set Grid Spacing” on page 483.

X Offset, Y Offset

X Offset and Y Offset set a known grid point the specified distance from the cell origin in the X- and Y-directions respectively. Use the offset to align the router grid with another grid, typically the one used for standard cells. See “Router Grid Offset Field” on page 484.

These optional values should match the DRC value for the process. However, the SDL router will determine the appropriate wire widths based on the size of the ports in the layout, as described in “Wire Width Computation” on page 485.

Route all traces in minimum width

When checked, traces will all be routed using the minimum routing width, without variation. See “Wire Width Computation” on page 485.

Enable bus routing optimization

When checked, the router will attempt to detect buses and route them together rather than treating them as independent signals.

Note: This option should not be enabled for standard cell routing.
**Set Grid Spacing**

You can specify a grid spacing, or let the router calculate grid spacing based on the router setup. It will calculate a spacing to provide the least constraints based on design rules for the design.

The best practice is to adopt grid and port settings from the standard cells. If you prefer to set the grid manually to adhere to an existing spacing domain, adhere to the minimum spacing formula shown below to prevent router failure.

\[
\text{MinGridSpacing} = \text{MinWireWidth} + \text{MinWireSpacing}
\]

**Router Grid Spacing Field**

When the **X Spacing** and **Y Spacing** fields are set to 0, the router analyzes the user-specified layer spacing, layer width, and via cell sizes for each routing layer and automatically computes a grid spacing.
based on the largest of all routing layers. Computed grid spacing for each routing layer is based on a wire-to-via spacing as shown below.

When a non-zero value is entered in any of the X Spacing, Y Spacing, X Offset, and Y Offset fields, the router also uses the layer spacing, layer width, and the size of the via cell specified by the user, but it will also determine whether to use via-to-via spacing or via-to-wire spacing rules. If the resulting grid settings are too small to support the specified layer width, spacing, and via cell, L-Edit displays an error.

Router Grid Offset Field

Each I/O port must be placed on a routing layer for the SDL router to connect to it, and they should be placed near the edge of the cell or where they can be easily accessed with routes. If they are placed too close to one another, the SDL router may not be able to connect to all the I/O ports.

Ideally, the ports will be placed on a specific routing grid so that the SDL router can directly access the ports without having to create breakouts. (See “Breakouts and Notches” on page 491.)
When you have cells that use a particular grid, you can set the router grid to match so that all ports are centered on a grid point using the spacing and offset fields in **Setup > Router**. Once the grid spacing is set, enter the X- and Y- offset amounts that will align the cell grid points with routing grid points.

![Grid Points Diagram]

**Wire Width Computation**

The width of routing geometry is subject to a minimum width constraint in the setup dialog but will be computed based upon the type and dimension of the port that it will connect to. When a connection is to be made between two ports with different dimensions, the smaller of the two trace widths is used for that connection.

For example, if a point port is to be connected to a line port, the minimum trace width defined for the layer in the **Setup Router** dialog will be used.

For box ports, the smaller dimension of the port will determine the route width. If the smaller dimension is less than the minimum specified route width, the minimum route width will be used.

The trace width computation applies the same way to both routing layers and non-routing layers. The table below describes how the trace width will be determined for each port type:

<table>
<thead>
<tr>
<th>Port Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point Port</strong></td>
<td>Uses the minimum trace width defined for the layer in the <strong>Setup Router</strong> dialog.</td>
</tr>
<tr>
<td><strong>Line Port (1D)</strong></td>
<td>Uses the dimension of the line port as the trace width. If the dimension of the line port is less than the minimum trace width defined for the layer in the <strong>Setup Router</strong> dialog, the minimum trace width will be used instead of the line port dimension. See for routing wide metal to line ports.</td>
</tr>
</tbody>
</table>
Routing Wire End Styles

Routing wires will be created using the Extend end style and the Layout join style. Note that when the router is attempting to connect to manually placed routes having a wire end style other than Extend, those wires will be converted to polygons. A breakout polygon, however, connects flush with a port.

Wide Wire Routing

The minimum port size for wide wire is the grid spacing plus the wire width.

\[ \text{MinPortLength} = \text{GridSpacing} + \text{WireWidth} \]
In order to auto-route traces with wide metal, one port dimension must be equal to or greater than the desired trace width.

**Locating Ports**

Ports should be located on the cell boundary for optimal routing. For connections inside the cell or within the obstruction layer it is recommended you manually route to place a port to the boundary rather than trying to connect inside the cell.

In some cases it may be simpler to route using a hole in the obstruction layer. For example, if you have a cell with crowded routing on layer M1 and you need to route to a port on M1, you can cut a hole or channel in the M1 obstruction layer, of M1 via width minimum, so that routing will come in to the port on layer M1.

If a port is on a layer that is not defined as a routing layer, it will be ignored.

To transfer the I/O ports that exist on the top level of a design in S-Edit to the SDL-generated top level in L-Edit, add one more level of hierarchy to the netlist. Or, you can run the auto symbol generator in S-Edit, then instance this symbol in a dummy toplevel.

**Trace to Port Connection Behavior**

Port to trace connection geometry will differ according to whether a routing or non-routing layer is used and whether the connection is to a point, line or box port.
Point Ports

Connections to a point port use the minimum trace width defined for the layer in the Setup Router dialog. The trace will be routed to a point port such that the center of the endpoint of the wire will exactly hit the port, as shown below.

If a port is off-grid, breakout routing is created to get the routing onto the grid. If the port is on a routing layer, the breakout will be on the same layer as the port.

If an off-grid point port is on a non-routing layer, the breakout will be on the next higher routing layer.
Line Ports (1D)

A trace will be routed to a line port such that the line is approached in a direction opposite its axis and contact is made approximately along the length of the line port, as shown below.

Box Ports (2D)

The router will assume that the box port is filled in the subcell with the same layer used to place the port and therefore may only connect to the edges of the box port. The trace will be routed to a box port based upon one of three scenarios as follows.

Note that routing to a box port will fail if the port is smaller in either dimension than the minimum trace width.

Box Port Larger than Trace in Both Dimensions

If both the length and width of the box port are greater than or equal to the minimum trace width, the trace may approach the box port from either axis and will contact the port along an edge equal to the width of the trace:

This dimension must be less than the minimum trace width.
Box Port Larger than Trace in One Dimension

[9] If one dimension of the box port is smaller than the minimum trace width and one is larger, the trace may approach the box port only along the longer axis:

Default Trace Width

Box Port

Box Port Smaller than Trace

[10] If both dimensions of the box port are smaller than the minimum trace width, the port is ignored. (You will see an error code in the .)

Obstruction Regions

An obstruction region is an area where the router cannot route. A unique obstruction layer may be assigned to each routing layer, or they can all be assigned the same obstruction layer. You are not required to set a obstruction layer. If there are keepout layers in instances, however, the router settings should match. You will need to use obstructions carefully to accommodate your routing layers, via design and wire end styles.

Note that because of grid placement requirements, a routing wire may sometimes overlap an obstruction boundary. See “Wire Width Computation” on page 485”.

Routing obstruction layers can be drawn in library cells or specified in the SDL top-level cell.

Note that obstruction regions do not adhere to DRC spacing rules, so metal can be drawn adjacent to an obstruction.

DRC rules do not apply - metal can be drawn flush with an obstruction.
To prevent routing failures, make sure your obstruction includes room for wires to connect to line or box ports in your design. If, for example, a design uses line ports, confine routing to a layer that does not have obstructions, manually route to the cell boundary, or draw obstructions so the ports are exposed.

Similarly, as shown in this example, if Poly is not a routing layer you can place a Poly-M1 Via on Poly obstruction to reach a port on Poly.

In some case during automatic routing using technology and cells from LEF/DEF files, the router can connect ports even if they are in obstructions. This occurs when the port distance from the edge is less than the sum of the metal width and the minimum spacing from the edge of the obstruction.

Vias also can be placed by the router even if the port is covered by the obstruction if the layer connecting is not blocked. However, if the port dimensions are smaller than the size of the via, the router does not place the via.

**Breakouts and Notches**

The router creates short pieces of non-wire geometry to connect off-grid ports to the on-grid wiring, or so that the routing adheres to DRC rules. These segments are called **breakouts**.
The router will create breakouts as required to complete connections if, for example, the ports in your cells are not designed on-grid, if the cell grid is offset from the routing grid, or if the cells are not placed in the layout so as to align with the routing grid.

Breakout placement and configuration will differ according to the port type (point, line, box), orientation, layer and via usage, trace width, and the presence of any obstructions or routing boundaries, as outlined on the following pages.

During auto-routing, L-Edit fills notches to close gaps that would otherwise be left, for example, between trace connected to a larger dimension via.

---

**Netlist Format for Assisted Routing**

The netlist can be a hierarchical SPICE or a structural Verilog netlist. Each device that is listed in your netlist should be a cell with pre-defined layout or a T-Cell prior to running assisted routing. The device name in the netlist must match the cell name in your TDB file for that device to be instantiated during generation from your netlist. Cells that do not exist or do not match by name may be optionally generated upon importing of the netlist for later layout.

If a subcircuit exists in the netlist with the same name as the currently active layout cell, then that subcircuit is used to create the netlist for the current cell. This behavior means that you can have all netlists for all cells in the same SPICE file. If a subcircuit with the same name as the current cell does not exist in the SPICE file, then the top-level SPICE entities are used to create the netlist.

Assisted routing orders the placement of instances by device group and then order of appearance in the SPICE netlist. Initial placements will occur within the MBB of existing objects or, if no objects exist, in a vertical stack.

The router will connect to a random initial placement of top level ports on routing layers.

Note that single quotes are reserved for arithmetic expressions that need to be evaluated. To pass string-valued parameters use the syntax PAR=str(“string value”) or PAR="string value”.

In the SDL Navigator, port names, pin names and instance names are matched in a case-insensitive manner. Cell names that are read from the netlist are matched to existing L-Edit cells also in a case-insensitive manner.
Implicit Model Definitions in Assisted Routing

When a device statement includes a model name, assisted routing looks for a corresponding .model statement in the netlist.

In SPICE, BJTs and other devices can have three terminals or four and they also need a model. In the case of a three terminal device, assisted routing is not able to determine if the fourth pin is a fourth terminal or a model, unless it has already encountered either that terminal nor that model before. When it has encountered a previous definition on another device, assisted routing will attempt to resolve this ambiguity, in which case it gives a warning “Implicit model definitions for yyy” in the assisted routing log. If the .model statement is missing, L-Edit will warn with the error message “internal error.”

To eliminate the need to write a model definition for BJTs and other devices, assisted routing can automatically add a simple model definition that contains no parameters (e.g., “.model mydevice”).

assisted routing will provide implicit model definitions for diode (D), JFET (J), GaAsFET (B), BJT (Q), MOSFET (M), and MESFET (Z) device statements. Such model definitions appear as warnings in the SDL output. For example, auto-declaration of the MOSFET model M1 would result in the following warning:

Warning: test1.sp(4): Implicit .model definition M1

Remapping Model Names

Assisted routing supports the .malias command, which allows L-Edit to recognize two different model names as the same model. See “.MALIAS” on page 1000 in the LVS section for more details.

Alternative Netlist Format

If you do not have a SPICE netlist available, the SDL Navigator can import or export a simple wirelist format, in which each line contains the following tab-delimited fields:

NODE NAME \t CELL NAME \t INSTANCE NAME \t PORT NAME

For I/O ports, the syntax is similar, but the cell and instance names are omitted:

NODE NAME \t \t \t PORT NAME

In both of these cases, spaces are shown around the tab character (\t) for clarity; these spaces are not present in the actual wirelist.

To import this wirelist format, select “WRL” from the File format pull-down menu in the Import Netlist dialog.

Bus Routing Optimization

Bus routing optimization automatically detects busses, identified as a set of ports in alignment (equal x or y coordinate) forming a connection to another set of ports in alignment, connecting in the same order. The first port in the first set connects to the first port in the second set, and so on. Ports must be point ports, line ports with width less than two wire widths, or rectangle ports with shorter side less than two wire widths. Bus routing optimization will attempt to route this collection of nets together, although there may be a transition region near the ports.
Running the Automatic Router

To run the automatic router, invoke **Router > Route all** from **SDL Command Menu**. To route selected nets, first select nets in SDL Navigator, then invoke **Route** from the context sensitive menu in SDL. The colored checks are used by the autorouter to indicate that a net is routed to completion. A red X (\(\times\)) indicates that all or a portion of the net could not be routed. A green check (\(\checkmark\)) indicates the entire net was routed successfully. The SDL Connectivity checker can be used to validate the connectivity of routing. The connectivity checker also updates the colored check marks.

**Using the SDL Navigator Checklist**

The navigator checkboxes are provided for the user to indicate to the SDL tool when a net has been manually routed to completion, and the net should be locked in place. A check on a pin name is used to indicate that the pin has been routed to, and a check on a net is used to indicate that the entire net has been routed. The Route All and Route selected commands will therefore not attempt to route nets that are marked with a check. Nets may be ripped up by unchecking those nets you wish to remove, and then invoking the Ripup Unmarked Nets command. Nets may also be ripped up by selecting nets in the SDL Navigator and invoking Ripup Net from the right click menu. This command will remove geometry tagged with the selected net names, and will also uncheck the checkbox for the selected net.

If you expand the symbol to the left of the net segment in the SDL Navigator, you can view the pins associated with that net segment, shown for node Vb1 above. You can then use the **Flyline** icon to show the connections that need to be made. For example, in the illustration above, flylines for Vb1 are drawn in red.

When you are viewing **By Net**, a middle-click on a given net will set it as the active net (causing the SDL Navigator to display the netname in italics.)
Standard Cell Placement and Routing

The Tanner place and route tool uses two steps to read in one or more LEF layout exchange format files for the technology settings and cell definitions used in standard cell placement and routing.

The first step imports routing layer and via cell definitions. In the second, L-Edit reads a LEF file for physical geometry - cell libraries, blockage (keepout) areas, port descriptions and routing geometry. You have the option to instance in cell contents from an existing cell.

Typically, LEF files are provided by a foundry as part of their process definition kit (PDK) or by a standard cell library vendor or designer. If you also have a DEF file with placement and routing information L-Edit can simply read it in.

Once L-Edit has the necessary technology settings and layout components, you designate a top-level cell with the boundary inside which the standard cells are placed and populate it with rows of the site cell that ordinates cell placement.

If you have a clock route you should add the buffers first, otherwise you can proceed to adding rail filler to the rails and then routing the remaining nets. You also have the option to route all nets at once and to route using just a single layer.

Import LEF Technology

Using the Command Menu button in the SDL Navigator ( ), select **LEF/DEF > Import LEF Technology** to choose a LEF technology file from which to import routing layers, widths and spacing, and preferred routing direction.

From File

Choose the Library Exchange Format (LEF) file with the routing, via and process definitions you want to use.

Target

Select from the open libraries the one to which LEF information will be imported. Most often this will be the “tech” or “technology” library.
Confirm Routing Layer and Via Definitions

After the technology data is imported, you can confirm that it is complete and accurate in the **Tech Layers** tab of **Setup > Design**.
Use the **Valid Vias** tab of **Setup > Design** to view the imported via definitions:

![Setup Design Interface](image)

**Via Name**
Vias are imported as custom vias.

**Lower Layer, Upper Layer**
Indicates the layers a via connects.

**Pitch X, Pitch Y**
Sets the distances between the origin of via cells in the X- and Y-directions respectively.

**Import LEF Cells**

You import port, keepout and cell content definitions from a LEF file using **LEF/DEF > Import Cells from LEF** from the Command Menu in the SDL Navigator ( ). You can instance cell contents from
existing cell views in the database. You can also choose to overwrite cells having the same name and view type.

![Import Cells from LEF Dialog Box](image)

**From File**
Select the LEF file containing the cell definitions you want to use.

**Target Library**
Indicate the library into which the LEF cells will be imported.

**Target View**
Indicate the view type the LEF cells should be assigned.

**Overwrite existing**
Overwrites cells in the database with the LEF cell when they have conflicting cell view names.

**Instance layout cell**
Check this option to instance another view from the same cell into the corresponding cells from the LEF import process.

**View**
Select the view type that will be used if you do instance another view into imported cells.

**Read or Write LEF/DEF Files**

You can use LEF/DEF from the Command Menu button in the SDL Navigator ( ) to read a DEF file (using Read DEF) or to export a LEF or DEF file (using Write LEF or Write DEF). DEF files are written and read without a dialog interface.
To write a LEF file you need to make some basic choices in the dialog shown below.

![Write LEF dialog](image)

**To File**

Enter a name and location for the LEF file that is written.

**Cells to write**

Use this section to choose which cells will be written to the LEF file. Options are the **Active cell**, **All cells in library** - in which case you must select a specific library, or from checked cells in the SDL Navigator using **Cells selected in Cell navigator**.
Configure the Placer

Before you initiate automatic placing, use the SDL Navigator command **Placer > Setup Placer** to ensure the placer will be using the correct files and settings.

**LEF files**
Confirm that the placer is referencing the desired LEF files. Note that the tech LEF file must be the first in this list.

**Additional items**
Use this field to insert commands that are written to override default settings.

- **density** \(d\), \(0 < d < 1\) default = 0.75
  Controls how closely cells are placed, where a higher value is more dense.

- **effort factor** \(e\), \(0 < e \leq 1\) default = 1.00
  A smaller value instructs the placer to process more quickly with less placement optimization over fewer iterations. A larger value instructs the placer to use more placement optimization with more iterations and less initial fixity of placement.

- **horizontalpreference** \(h\), \(0 < h < 1\) default = 1.00
  When this value is greater than 1, instructs the placer to conserve vertical routing resources by reducing the vertical distance between pairs of connected cells.

  When this value is less than 1, vertical routing is preferred. Pairs of connected cells are placed more closely in the horizontal direction to preserve horizontal routing resources. So, -horizontalpreference 1.5 and -horizontalpreference 0.66 are comparable except for the preferred direction used.
Choose a Top-Level Cell and Add Site Cells to Define Spacing

The cell that is active when you initiate placement is the “top-level” cell into which the standard cells are placed. Typically it will contain just a boundary box or polygon.

The first step in placement is to fill this boundary with an array of a special cell, the site cell, to create a row configuration the placer uses to position standard cells.

The site cell is designated by the keyword SITE in the LEF file. If the proper cell view has not been imported from the LEF file you must create your own site cell.

With the top-level boundary cell active, use the SDL Navigator command Placer > Fill shape with rows to select your site cell.
Rows are placed in the top-level cell beginning with a site cell at the origin (shown below in yellow.) Site cells snap to a grid of their own size regardless of the boundary, so that all four corners of any site cell fall within the boundary. They will not, for example, be rotated or truncated to fit within the MBB.

Launch Automatic Placement

Once standard cells are in your library you can proceed to automatic placement. Launch automatic placement with the SDL Navigator command **Placer > Run Placer**. The results will look similar to the example below. Any errors are written to a `write_def.log` file.
Additional Placer Commands

You can export a CDL format netlist or view the ports on the current cell with these commands.

**Write SDL Netlist to CDL**
Opens a standard “Save As” window so you can export a netlist of the design for LVS purposes.

**Write SDL Netlist to Verilog**
Save the netlist to a file in Verilog format

**Show Interface**
Opens a “cell interface” text file that shows just the inputs and outputs of the current cell.
Add Clock Net Buffers

You can use the SDL Navigator command Clock Router > Insert Clock Buffers to specify one or more clock nets and the correct buffer cell. Clock routing attempts to minimize skew on the net by balancing net lengths and inserting buffers to increase signal propagation speed to more distant receivers.

Clock nets
- List one or more clock nets that require a buffer.

Clock buffer
- Choose a buffer cell and view from a library that is open.

Max fanout
- Enter the maximum number of standard cells that can be driven by this buffer cell (must be greater than or equal to one).

Note that adding buffers may split the clock net into subnets as shown below.
Add Fill

If you hide all but the routing layers after placement you can more easily see where the ground and power rails are not continuously connected.

Use **Placer > Insert Filler** to close those gaps with instances of the filler cells.

**Place Power Rails**

You can place Power and Ground rails using the SDL Navigator command **Placer > Place Power Rails**

![Power Routing](image)
Power rails can be placed on the left, right, or both, as shown below.

**Route Clock Nets**

Use the SDL Navigator command **Router > Setup Router** to enter routing configuration choices (see “Configuring the Automatic Router” on page 479).

Use **Clock Router > Route Clock Nets** to list the clock nets you want automatically routed.

If you are not satisfied with the clock net routing you can use **Clock Router > Ripup Clock Nets** to remove one or all of them, and then repeat the clock routing process.

**Route Remaining Nets**

Launch the autorouter using **Router > Route All** to route all cells or **Route all only on** to route on just one of the routing layers. The **Open Router Log** displays a text list of the nets that were successfully routed.
Sample Standard Cell Placement and Routing Tool Flow

Library setup, done once per library

[1] Begin with a technology setup library

[2] Create Standard Cell Library by importing the GDS for the library, or attach existing Standard Cell Library

[3] Import LEF technology by invoking SDL Command Menu > LEF/DEF > Import LEF Technology, and choose the LEF file that contains the technology for your process. This will setup the routing layers, width, spacing and preferred direction for each layer, via layers in the Setup Design > Tech Layers dialog. Obstruction Layers and Pin layers should also be specified in Setup Design > Tech Layers, if other than default. Also, purposes for placing new objects should be specified in Setup Design > Tech Layers, if other than default.

[4] Import LEF standard cells by invoking SDL Command Menu > LEF/DEF > Import Cells from LEF, and choose the LEF file that contains the standard cells for your process. This creates a new cell, with name equal to the macro name in the lef file, in which the ports and keepout areas for each standard cell are defined, and optionally instances the cell containing the full layout. In the Import Cells from LEF dialog, setting the Target View equal to LEF and checking “Instance layout cell” will create a standard cell with view name LEF, and the GDS layout cell instanced in the LEF view. Choose “layout” for the Instance layout cell view, if your GDS layout views names are “layout”.

Placement and Routing, done for each logic block

[1] Create a cell for the place and route. If using a Verilog netlist, the cell name must match the module name in the Verilog file.

[2] Import the synthesized netlist into SDL. The options should be set to add ports (“Add missing I/O ports”) but not instances. The placer will add the instances.

[3] Draw a box or polygon for placement. The cells will be placed inside this shape. This may be any shape, but needs to be large enough for your total cell area divided by your fill factor.

[4] Setup the placer by invoking SDL Command Menu > Placer > Setup Placer (add your tech lef and standard cell LEF, tech must come first)

[5] Fill the placement shape with rows of site cells by invoking SDL Command Menu >Placer > Fill shape with rows (use the LEF version of your SITE cell)

[6] Run the placer by invoking SDL Command Menu > Placer > Run Placer

[7] Insert clock tree by invoking SDL Command Menu > Clock Router > Insert Clock Buffers


[10] Check the router grid settings by invoking SDL Command Menu > Router > Setup Router. Make sure that it EXACTLY corresponds to the ports in the standard cells. Any error in the size or offset and routing will run forever or fail.

Checking Connectivity

You can identify and locate short and open circuits, unknown circuit elements or missing nets and instances by loading a new netlist then using **SDL Command Menu () > Extract > Check Connectivity** to verify SDL connections. Results will be displayed in the Verification Navigator.

At this time the Connectivity Checker supports boxes and wires, but does support support polygons.

Writing SDF

A Standard Delay Format (SDF) file may be written by invoking **SDL Command Menu () > Extract > Write SDF.**

![Write SDF](image)

The SDF file is a calculation of driver to receiver interconnect delay. L-Edit calculates a slow, typical, and fast delay for rising and falling edges for the interconnect between each driver and receiver. Rise and fall impedance and capacitance values are obtained from three input liberty files containing the slow, typical, and fast driver characteristics. All three liberty files are not required, and L-Edit will calculate values corresponding only to those files given, if all three are not provided.

Writing SPEF

A Standard Parasitic Exchange Format (SPEF) file may be extracted using the Parasitic Extraction (PX) engine. To do this, invoke **SDL Command Menu () > Extract > Extract SPEF Using PX.**

To use **Extract SPEF Using PX**, LEF Technology and LEF cell definitions must be specified in the Setup Placer dialog, and **Setup Design > Tech Layers** must be configured.
Introduction

Interactive DRC lets you create highly compact designs by highlighting DRC violations in real time as you draw and edit.

Interactive DRC shows a colored outline when a polygon violates one of the defined rules (that is, when a minimum layout distance is exceeded), and optionally, another colored outline when a violation limit is reached. Markers display while an object is being drawn or edited but not after it has been placed.

The following rule types are supported:

- “Width” (page 512)
- “Spacing” (page 512)
- “Surround” (page 513)
- “Overlap” (page 513)
- “Extension” (page 514)
Setting Up Interactive DRC Rules

To setup interactive DRC, invoke **Setup > Design**, and select the **Interactive DRC** tab, or click the **Setup Interactive DRC** toolbar button.

**Enable Interactive DRC**
Enables the interactive DRC function when checked.

**Prevent violations**
Prevents you from moving an object past the first violation distance it encounters.

**Show distances**
Check this box to show the distance between the orthogonal edges of the active object and other objects that are closest to or most in violation of any rule. If the edge pair closest to or most in violation in non-orthogonal, a single diagonal ruler will display.

**Enable All**
Enables all interactive DRC rules.

**Disable All**
Disables all interactive DRC rules.

**Import from DRC Standard**
Imports the built-in set of Tanner design check rules. See “DRC Rule Sets to run” on page 548.

**Violation Marker Color**
Use these controls to set the color of the interactive DRC ruler and edge marker when an object is in violation.

- **Auto** sets the color of the violation marker to the last color in the color palette, usually black.
- **Pick** lets you choose the marker color using the standard **Color** selection dialog.
In the illustrations below, the limit marker is cyan blue and the violation marker is red. In the first example, the via is at the .150 micron limit for Poly layer surrounding the Contact layer.

When the via is moved to the right and up, it falls into violation of the Metal1 to Metal1 spacing rule of .350 microns.
Width

Width rules specify the minimum width of all objects on the specified layer. The width measurement is not displayed when an object is being moved since its edges are not changing with respect to one another.

Spacing

The spacing rule specifies the minimum distance that should separate all pairs of objects, either on the same layer or two different layers.

Objects touching by coincident edges, intersecting objects, and one layer enclosing the other layer are not flagged as violations by interactive DRC. Also, edges intersecting at an acute angle are not flagged as a violation.
**Surround**

The Surround rule specifies that objects on one layer must be completely surrounded by objects on another layer.

Inside layer polygons completely outside the surrounding layer are not flagged as violations by interactive DRC.

**Overlap**

Overlap rules specify the minimum amount that an object on one layer must overlap an object on another layer (when there is an overlap).
Objects which overlap more than the specified distance or whose edges coincide are not considered in violation of overlap rules.

**Extension**

Extension rules specify the minimum amount that an object on one layer must extend beyond the edge of an object on another layer.

One layer completely outside the other layer, or one layer completely inside but not inside coincident with the other layer is not considered a violation.
Running Interactive DRC

To enable Interactive DRC click the Enable Interactive DRC toolbar button. After enabling Interactive DRC begin editing your layout. Error markers will appear in the layout as soon as violations are detected.

Importing Interactive DRC Rules

Tanner EDA provides the ability to import interactive DRC rules for customers who are doing verification using DRC decks provided by the foundries. It is possible to import Calibre, Dracula, or Assura files.

Alternately, if you use Virtuoso Import, for those tech files that have the rules, it will create standard DRC rules for drawn layers with spacing, width, and surround information in the tech file. You can then use import from DRC Standard on the Interactive DRC Setup tab to automatically set up interactive DRC rules.

Below is the translation that L-Edit uses to convert the five rules that interactive DRC supports. Note that the presence of the INSIDE ALSO option is used to determine whether ENC translates to "Surrounding" or "Extension out of."

///// INT Rules /////

Width { INT L1 < dist }
// Translation: Width L1 dist

Overlap { INT L1 L2 < dist }
// Translation: L2 Overlap into L1 dist

///// EXT Rules /////

SingleLayerSpacing { EXT L1 < dist }
// Translation: L1 Spacing L1 dist

TwoLayerSpacing { EXT L1 L2 < dist }
// Translation: L2 Spacing L1 dist

///// ENC Rules /////

Surrounding { ENC L1 L2 < dist INSIDE ALSO }
// Translation: L2 Surrounding L1 dist

Extension { ENC L1 L2 < dist }
// Translation: L2 Extension out of L1 dist

Use Setup > Design > Interactive DRC > Import from file to begin the import. The translation adheres to these constrains.

- The input stream of commands is scanned, and only a subset of commands are actually accepted. Commands that do not fulfil some criteria are silently ignored. Currently we import INT, EXT (1 and 2 layers) AND ENC (with or without INSIDE ALSO)
- Various Calibre options change the commands semantics to the extent that the translations should either be changed or the commands be entirely skipped. L-Edit checks for 'NOT' and skips the command if the option is present. All other options are ignored.
Layer matching happens by GDS number, not by name.

In Calibre, only commands containing exclusively drawn layers are accepted.

A simple optimization ensures that when two completely identical commands are imported, only the one with the minimal distance value is accepted.

Upon importing, the data goes to the dialog and is not stored to the database until OK is pressed.

Unlike “import from internal,” the old commands are not removed. This allows importing multiple files, one after another.
15 Node Highlighting

Introduction

Node highlighting allows you to highlight the geometry connected to a selected net. An unlimited number of nets may be sequentially selected. When you have selected an object from lower in the cell hierarchy, all related connectivity from the top level cell will be displayed. If a selection touches more than one net the potential nodes are displayed and you will be prompted to pick one.

To highlighting nodes you must first define connectivity statements using the Setup Design dialog, then extract the connectivity data and select nodes for highlighting either using the mouse or by typing in node names.

Node Highlighting Setup

Choose Setup > Design and select the Node Highlighting tab, or click the Node Highlight Setup toolbar button to enter connection statements. You can enter connection statements manually, or import them from an extract definition (.ext) file.

Node highlighting works on merged objects on drawn or derived layers. If derived layers are used, they must be set up in the TDB file before connectivity data can be extracted. (The node highlight engine will generate any needed derived layers.)

Objects are defined as connected if the AND of objects on Layer1, Layer2 and the connection layer results in geometry. Objects that touch are not considered connected. If a connection layer is not specified, Layer1 and Layer2 must overlap to be considered connected.
Layer1, Layer2
The layers that are connected by the connection layer. Layer1 cannot equal Layer2.

Connection Layer
The layer through which Layer1 and Layer2 connect. If either Layer1 or 2 is equal to the Connection Layer value, the Connection Layer should not be assigned a value.

Auto
Sets the color of the node marker to the last color in the color palette, usually black.

Customize
Opens the Node Highlighting Colors dialog, which allows you to enter an unlimited number of colors for the Cycle function. Each color can be customized using the RBG sliding controls or by clicking on Color picker... for advanced color controls.

Default sets the color cycle to eight pre-selected colors.

Cycle
Cycles through the colors defined in the Node Highlighting Colors dialog as nodes are selected for highlighting. The color cycle starts with either the custom or automatic color depending on which of these options is checked.

If this option is not checked, all nodes are highlighted with the same color.

Import from EXT file...
Imports the connectivity information from an extract definition (.ext) file.
Chapter 15: Node Highlighting

Node Highlighting Setup

Default Location of Saved Node Highlighting Data

When you extract connectivity, L-Edit creates a NodeHighlighting folder with a [designname_nh] subfolder as the default location for data from each cell that has node highlighting data.

It also activates the “enable node highlighting” button, which locks those data files so they cannot be opened elsewhere at the same time.

...\NodeHighlighting\test_lights_nh

The Node Highlighting Toolbar

You can run node highlighting using the menu commands in Tools > Node Highlighting or the Node Highlighting toolbar. The table describes the icons below from left to right:

1. **Save Folder...**
   - Use this button to save connectivity data to a folder other than the L-Edit default location, which is described in “Default Location of Saved Node Highlighting Data,” below.

   **Note:** The Setup Extract—Options dialog and the Setup > Design—Node Highlighting dialog each include the option to save node highlighting data, and to choose where it is saved. Be careful to manage these settings to avoid losing or overwriting your data.

2. **Node Naming**
   - Sets the level of hierarchy from which ports will be used for node naming during node highlighting. Choose from None, Top level only or All levels, or Depth. None will be the fastest setting, All levels the slowest.
   - None—no ports are written to GDSII for node highlighting.
   - Depth—enter an integer value to specify the level of hierarchy that will be used. A depth of zero is equivalent to top level only.

   Note that the Highlight by name command can be used regardless of the Node Naming setting.

3. **Extract Connectivity**
   - Runs the connectivity extraction engine and saves the results in the design directory. Connectivity data is saved with the design file. If connectivity information already exists, you will be prompted before the data is extracted again.
Highlight Node

Initiates node highlight mode. Once active, a left mouse click highlights a node.

Each merged polygon on each of the node layers will be highlighted, and the node name will be displayed in the status bar. Selecting a node that is already highlighted will give you the option to remove that highlight or change its color.

Previous node highlights remain when a new node is selected.

Highlight by Name

Opens the Highlight Node by Name dialog which allows you to highlight a node by name. Matching is case-insensitive. If the node name is not found or if it matches more than one node, a warning will indicate so and advise on how a node will be selected if it is not unique. A history drop down list is provided for the Node Name field.

Zoom to Node

Pans and zooms to display all highlighted nodes in entirety plus a 10% margin.

Toggle Markers

Toggles the display of all design markers (node highlighting, DRC, SDL, cross probing, etc.).

Clear Markers

Deletes the display of all design markers (node highlighting, DRC, SDL, cross probing, etc.).

Enable Node Highlighting

Use this button to obtain and release the node highlighting license. (It will appear lighter when enabled.) Node highlighting data is license controlled so that it cannot be opened by someone else when it is in use. See “Default Location of Saved Node Highlighting Data” on page 519.

Disabling node highlighting clears all markers as if you had used the Clear Markers button.

Node Highlight Setup

Opens the Setup Design dialog at the Node Highlighting tab. See “Node Highlighting Setup” on page 517 for further information.

Open Connectivity Extraction Log

Opens a text window displaying the Connectivity Extraction Log. The log lists the port name and coordinates (in internal units) of ports found on the same net. The log file is created in a temp directory and is erased when the TDB file with which it is associated is closed.

Data is described in the following format, where the net number is randomly assigned:

```
Found port conflict in net 30358248 within cell Core:
U33/Mux2_15_2/Out at (1216000, -420000)
U33/Mux2_8_1/A at (1136000, -420000)
```
Using Node Highlighting

[1] Use Tools > Node Highlighting > Extract Connectivity or the button to extract connectivity data. If connectivity data for the cell already exists L-Edit will confirm that you wish to replace it.

[2] Once you have extracted connectivity data, you can highlight a node either by name or by clicking in the layout. Using the mouse, pick Highlight Node then click on a point in the layout.

[3] If you pick a point that belongs to more than one node, L-Edit will list those nodes so you can choose the one you want. When two nodes are highlighted in the list it often means that there is a sublayer.
[4] If you pick a node that is already highlighted, L-Edit will prompt you to remove or change the color of the existing highlighting or to cancel the node selection process.

![L-Edit Message]

[5] To highlight a node by name, click on **Highlight by Name** and type the name of the node you want.

![Highlight Node by Name]

[6] L-Edit will highlight nodes using the marker color settings entered in the **Node Highlighting Setup** dialog.

![Highlighted Nodes]
It can be easier to see the highlighted nets when you toggle the markers on and off or if you hide most layers. In the example below, the lower image shows only the “Active Contact” layer, and you can see that the marker color has cycled from red to orange.

You can also press W to zoom into a marker.
16 Add-Ins

The **Tools > Add-Ins menu** contains a miscellaneous set of macros for relatively uncommon tasks.

Add-In Macros Documented in This Chapter

- "Area Calculator" (page 525)
- "Instance Location Summary" (page 525)
- "Mask Bias" (page 527)
- "Mark Cells for Flattening" (page 528)
- "Layer Fill" (page 529)

You can use the shortcuts **Tools > Macros** and **Tools > Repeat Macro** to speed access to your most recently used macros.

- "Macro" (page 531)
- "Repeat Macro" (page 531)

Add-In Macros Documented in Other Chapters

The following macros are discussed in the sections of the manual to which they relate functionally:

- "Transferring File Information to Cells" (page 123)
- "GDSII Properties" (page 158)
- "Exporting PostScript Masks" (page 153)
- "Finding I/O Pads in the Fabrication Cell" (page 267)
Area Calculator

This macro calculates the total area, the minimum bounding box size and the density of the top-level of a cell (ignoring instances).

Instance Location Summary

This macro generates an .htm format table that lists all cell instances in the active view. Colored values are used to highlight a translation point that is not on an integer increment of the micron grid.

<table>
<thead>
<tr>
<th>Cell Name</th>
<th>X (microns)</th>
<th>Y (microns)</th>
<th>Orientation</th>
<th>Scale</th>
<th>X Repeat</th>
<th>Y Repeat</th>
<th>Delta X</th>
<th>Delta Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>pnos</td>
<td>-77.750</td>
<td>90.550</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>pnos</td>
<td>-111.550</td>
<td>90.550</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>pnos</td>
<td>-145.350</td>
<td>90.550</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>pnos</td>
<td>-179.150</td>
<td>90.550</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>nmos</td>
<td>-111.450</td>
<td>61.950</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>nmos</td>
<td>-141.450</td>
<td>61.950</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>nmos</td>
<td>-171.450</td>
<td>61.950</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Cell Name

Names the cells instanced in the cell on which the macro ran.
<table>
<thead>
<tr>
<th><strong>X (microns), Y (microns)</strong></th>
<th>Cites the displacement in the x- and y-direction, respectively, of the instance with respect to its primitive cell.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Orientation</strong></td>
<td>Indicates the clockwise degree of rotation with respect to the primitive cell, where zero is no rotation and a negative value is in a mirrored position.</td>
</tr>
<tr>
<td><strong>Scale</strong></td>
<td>A value ≥ 1 indicates the instance has been enlarged, a value &lt; 1 indicates the instance has been shrunk, and 1 indicates no change in size.</td>
</tr>
<tr>
<td><strong>X Repeat, Y Repeat</strong></td>
<td>For instance arrays, indicates the horizontal (X) and vertical (Y) repeat counts of the instance.</td>
</tr>
<tr>
<td><strong>Delta X, Delta Y</strong></td>
<td>For instance arrays, indicates the X and Y spacing between array elements, measured origin to origin.</td>
</tr>
</tbody>
</table>
Mask Bias

Use this macro to performing mask resizing on a layer by layer basis. You can remove or merge geometry as well as shrink or grow objects by layer.

- **Drawn**: Use this list to select the Drawn layer.
- **Grow By (Lambda)**: Enter the number by which the layer(s) will grow (positive number) or shrink (negative number).
- **Mask**: Use this list to select the Mask layer.
- **Delete geometry on target layer prior to resize**: Deletes the geometry on the layer where the change will take effect, before the resizing operation.
- **Delete geometry on source layer after resize**: Deletes the geometry on the layer that does not change, after the resizing operation.
- **Merge geometry after resize**: Merges the geometry on the layer where the change is applied, after the resizing operation.
Mark Cells for Flattening

Use this dialog to select the cells that will be assessed for flattening during DRC, Extract and Node Highlighting.

Resize geometry in:

- **Active cell only**—applies the change to all geometry on the selected layers of the currently active cell.
- **Selected objects in active cell only**—applies the change only to selected geometry on the selected layers of the currently active cell.
- **Active cell and descendants**—applies the change to all geometry on the selected layers of the currently active cell and all its instances in the design.
- **All cells in active file**—applies the change to all geometry on the selected layers of the all cells in the design.

**Drawn - > Mask**

Sets the Drawn column as the source layer(s) and the Mask column as the target layer(s).

**Mask - > Drawn**

Sets changes to be applied to geometry on layers in the Drawn column with respect to geometry on the Mask layer(s).
Layer Fill

You can use **Tools > Add-Ins > Layer Fill** to add fill geometry to, for example, meet density rule constraints. This tool lets you place a predefined cell inside or outside any layers you select, or into the highest level of the cell hierarchy instead.

**Fill**

- **Inside source layer(s)**—fills the layers that are selected in the Source layers field.
- **Outside source layer(s)**—fills outside the layers that are selected in the Source layers field.
- **Top level boxes**—fills the geometry on the top level of the cell hierarchy.

**Source layer(s)**

Select the layers that are affected by the action you have set in the Fill field.

**None**

Use this button to deselect all layers in the Source layer(s) pane.

**Spacing to the fill**

Enter a value that will set how far, in microns, the fill needs to be from the edge of existing geometry in the cell being filled.

**Cell to fill with**

Choose the cell that will be used as a filler from the existing cells in the file. Usually you will have drawn a cell expressly for use as layer fill.
Layer Fill Options

Use these options to further control layer fill placement.

**Place fill into Cell**
When this box is checked, you can enter the name of a cell that will be created to contain the layer fill geometry, so that you can edit or delete it easily.

**Options**
Use this button to open the Layer Fill Options dialog (see below.)

Layer Fill Options

Use these options to further control layer fill placement.

**Place fill instances on grid**
Sets the grid spacing of fill instances, in microns.

**Do not fill instances on layer**
Lets you pick a layer inside of which fill instances are not placed.

**Do not fill instances of these cells**
Select cell(s) to exclude from the layer fill operation.

**None**
Use this button to deselect all cell in the Do not fill instances of these cells list.
Macro

Tools > Macro opens a dialog that allows you to run any of the macros loaded to its list, and to add the listed macros to the L-Edit Tools pulldown menu.

![Macro dialog]

**Macros**
- Highlight a macro in this list to Run it. Close closes the entire dialog.

**Macro Files**
- Load—lets you browse for new .c and .dll macro files to add to the list.
- Reload—lets you reload macro using the existing path and filename, for example when a file has been updated.
- Unload—removes a macro from the list.
- Setup—opens Setup Application at the UPI tab so you can specify the location of macro interpreter header files.

**Load Files at Startup**
- Saves the currently highlighted macro to the Tools menu beneath Add-Ins. Note that only one macro at a time will be saved.

Repeat Macro

Tools > Repeat Macro simply re-opens the dialog of the most recently used macro.
17 Introduction to Design Verification

Design Verification in L-Edit

This section of the *L-Edit User Guide* describes the design verification features of L-Edit:

- Standard DRC — easy to use design rule checker with setup dialog
- HiPer Verify — 64 bit design rule checker supporting Calibre, Assura and Dracula syntax
- HiPer Verify — parasitic extraction tool
- Extract — netlist extraction tool
- LVS — layout-vs.-schematic comparison application
- LVL — layout-vs.-layout comparison wizard

Chapter 18, DRC Standard Rules (page 533) describes L-Edit/DRC Standard. Standard DRC features user-programmable rules and handles minimum width, exact width, minimum space, minimum surround, non-exist, overlap, and extension rules. It can run a full-chip or region-only DRC. Error markers allow you to quickly and easily locate design rule violations.

Chapter 20, HiPer Verify: Calibre Command Files (page 557) and Chapter 21, HiPer Verify: Dracula Command Files (page 723) describe HiPer Verify. HiPer Verify features the ability to run Calibre, Assura and Dracula format command files without modification or translation. HiPer Verify features a larger set of available commands than DRC Standard, the ability to reprocess results with conjunctive rules, plus the ability to run DRC as a background process. It is fully integrated in the layout editor, and results are placed directly into an error navigator for viewing.

Chapter 24, Extracting Layout (page 912) describes Extract, the netlist extractor. Extract creates SPICE-compatible circuit netlists from L-Edit layouts. It can recognize active and passive devices, subcircuits, and the most common device parameters, including resistance, capacitance, length, width, area, and source and drain area.

Chapter 28, Introduction to LVS (page 971), Chapter 29, LVS Output Tutorial (page 1010), Chapter 30, Netlist Comparison (page 1022) and Chapter 31, LVS Command-Line Syntax (page 1030) discuss LVS, or *layout versus schematic*, a tool that compares two netlists to determine whether they describe the same circuit. When they do not, LVS works in conjunction with L-Edit to identify and correct errors.

L-Edit also provides a handy tool, layout versus layout (or LVL) which compares two L-Edit layout files for differences in their geometry. This feature is described in Chapter 23, Layout vs. Layout (page 904).

Syntax and usage for the file formats used in design verification are detailed in Chapter 28, Introduction to LVS (page 971).
Design Rule Sets

It is not usually necessary to create design rules sets from scratch. If you have a previous design file that uses a set of rules similar to those you want to employ in your current design, you can modify the rule set from the previous design.

In general, you must perform the following three steps to create or edit a design rule set:

1. Determine which rules must be specified. Fabrication services or foundries are typically able to provide design rule sets.

2. Determine which generated layers, if any, will be needed to implement each rule in the set. Define these layers using "Layer Setup" (page 89).


Setups

The design rule set is part of the setup specification that characterizes every L-Edit design. This setup should be established before you start any new design. Again, it is not usually necessary to create this setup from scratch. If you have a design file that uses the same or a set of rules similar to those you want to employ in your current design, you can modify the rule set from the previous design.

To use existing rules in a new file, you can either copy a previous design setup to the new file, or you can combine rules from different designs in a new setup. These two methods are described below.

Copying Setup Information to a New File

There are three ways to copy setup information to a new file:

- **Copy a design file.** The copy will automatically contain the same setup information as the original design file.

- **Create a new file while a file with the desired setup information is active.** The new file will automatically contain the same setup information.

- **Import from a TDB file.** Use the command **Setup > Import Technology** and select **TBD** from the drop-down list to copy the setup information from the specified file to the active file.

If necessary, you can modify design rules in the new or copied file using the Setup Design Rules dialog (see “Specifying DRC Standard Design Rules” on page 542).

Combining Rules from Different Files

You can combine design rules from different files as follows:
- Start by opening the design file into which you want to introduce additional rules. In that file, use the Setup Design Rules dialog to delete the design rules that you do not want to keep.

- Create any additional needed layers, including generated layers, and remove any unused layers. You should be careful in this step to create all and only the required layers. Design rules associated with missing layers will not work properly. No design rules will be specified for extra layers, so layout errors on those layers will not be detected.

- Specify the TDB file from which existing rules are to be taken by selecting select TBD from the drop-down list in the Setup > Import Technology dialog checking DRC rules in the Modules group box. Enter the name of the source file, uncheck all the options except DRC rules, and click OK. L-Edit reads the specified setup, including design rules, into the active file.

**Generated Layers**

You can specify design rules for generated layers just as you would for other layers. When you use generated layers in the specification of a design rule, L-Edit automatically generates objects on those layers then deletes these objects when the check is complete.

Following the DRC run, L-Edit automatically deletes objects on generated layers that were created during the DRC run. If Enable Derivation is off in the Setup Layers—Derivation dialog, then DRC doesn’t generate or delete the derived layer.

**Exporting DRC Standard Rules to Calibre Format**

You can export DRC setups in DRC Standard Rule format to Calibre format to use as a starting point for writing more advanced rule sets in text file format. To export DRC Standard rule to a text file, select Tools > DRC Setup from the menu, then highlight DRC Standard Rule Set and press the Edit button to open the Setup DRC Standard Rule Set dialog. Press the Export to Command File button.

**Running DRC from the Command Line**

HiPer Verify may be run from the command line as well as from inside L-Edit. Command line HiPer Verify will print documentation for the required syntax when you run hiperverify64 -help at the command line. Command line Verify takes layout in GDS format only, and command files in Calibre, Dracula, and TDR (Tanner Design Rule) format.

After running command line DRC, you can use the Import Results feature in the Verification navigator to import results back into the Navigator. The -c option is a useful way to avoid specifying full paths.

**Creating Precompiled Command Files:**

A precompiled TDR (Tanner Design Rule) file can be created from the user interface or from the command line.

From the user interface, right click on the rule file in the Setup DRC dialog, and choose Export as optimized rule deck (tdr) . From the command line, use hiperverify64 rulefile.cal -o rulefile.tdr.

The following parameters are available:

```
<rule file> <gds file>   Rule and gds file names, respectively. Rule file may be Tanner .tdr, Calibre, or Dracula
```
Design Rule Types

L-Edit supports seven types of design rules. Each of these rule types is described below, with specific examples shown in the accompanying figures.

- “Minimum Width” (page 535)
- “Exact Width” (page 536)
- “Not Exist” (page 536)
- “Spacing” (page 536)
- “Surround” (page 536)
- “Overlap” (page 537)
- “Extension” (page 537)
- “Density” (page 538)

Minimum Width

Minimum width rules specify the minimum width of all objects, in any direction, on the named layer.

Poly Minimum Width = 2 lambda

You can specify exceptions for this rule type as described in “Rule Exceptions” on page 538.
Exact Width

Exact width rules specify the exact width of all objects on the named layer.

Poly Contact Exact Width = 2 lambda

Not Exist

Not exist rules specify that no objects should exist on the named layer. Not exist rules are unique in having no associated distance.

Spacing

Spacing rules specify the minimum distance that should separate all pairs of objects, either on the same layer or two different layers.

Via to Poly Contact Spacing = 2 lambda

You can specify exceptions for this rule type as described in “Rule Exceptions” on page 538.

Surround

Surround rules specify that objects on one layer must be completely surrounded by objects on another layer.

Metal2 Surround Via = 1 lambda

You can specify exceptions for this rule type as described in “Rule Exceptions” on page 538.
**Overlap**

Overlap rules specify the minimum amount that an object on one layer must overlap an object on another layer (when there is an overlap). Objects which overlap more than the specified distance or whose edges coincide are not considered in violation of overlap rules.

![Overlap diagram](image)

Poly Contact Overlap Poly = 2 lambda

**Extension**

Extension rules specify the minimum amount that an object on one layer must extend beyond the edge of an object on another layer. Objects are not considered in violation of extension rules when they:

- Extend more than the specified distance
- Have a coincident edge but are otherwise outside
- Are entirely surrounded
Density

The density rule finds and flags objects on the derived density layer specified in **Layer1**. The layer specified must be a **Density** type derived layer. Violations to the rule include any polygons output to a density layer.

See “Density Layer Derivations” on page 457 for a description of density layers.

**Rule Exceptions**

You can fine-tune some rules by specifying particular layout conditions that are **not** to be reported as violations. These conditions are represented by the **Ignore** options in the dialog **Setup Design Rules** (see “Specifying DRC Standard Design Rules” on page 542).

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
<th>Applicable rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coincidences</td>
<td>Coincident edges between objects are ignored.</td>
<td>Surround</td>
</tr>
<tr>
<td>Intersections</td>
<td>Intersections between objects are ignored.</td>
<td>Surround</td>
</tr>
<tr>
<td>If layer 2 completely encloses layer 1</td>
<td>Objects on one layer <em>entirely surrounded</em> by objects on another layer are ignored.</td>
<td>Spacing</td>
</tr>
<tr>
<td>If layer 1 completely outside layer 2</td>
<td>Objects on one layer <em>entirely outside</em> objects on another layer are ignored.</td>
<td>Surround</td>
</tr>
<tr>
<td>Acute angles</td>
<td>Errors caused by acute angles (less than 90°) are ignored.</td>
<td>Minimum width, Spacing, Surround</td>
</tr>
</tbody>
</table>
The following illustration shows exception conditions for spacing and surround rules.

**Acute Angles**

When two consecutive edges of a polygon form an acute angle, the distance between them always goes to zero at the vertex. This means that acute angles, by definition, will lead to violations of minimum width and minimum spacing rules in DRC.

By checking the option **Ignore: Acute angles**, you can instruct L-Edit to ignore violations caused by acute angles. This option does not exclude objects with acute angles from the design rule check. Instead, it suppresses errors that result from an acute angle in the layout.

For example, consider the following all-angle polygon drawn on layer Poly:
Checking this object for minimum width would ordinarily yield three violations, as marked here:

Running the same design rule check with the option **Ignore: Acute angles**, however, ignores those violations that are caused by an acute angle. With these errors excluded, L-Edit reports only one violation.

**Flag to Append Special Commands**

When L-Edit runs a standard DRC rule, the DRC engine first converts the rule to Calibre format. You can use the special flag “|||” after a design rule to instruct the DRC engine to append any commands that follow the ||| characters to the Calibre rule.

For example, when **Display curves using manufactured grid** is checked, L-Edit will convert curved objects to polygons, with vertices that snap to the manufacturing grid. DRC treats these segments as separate objects, and if they cause a spacing violation will generate hundreds of violations.

While you can limit the number of errors reported for a single DRC rule, the limitation applies even when the errors come from different cell instances. Similarly, temporarily increasing the size of the manufacturing grid can decrease the number of DRC errors, but also changes the geometry to an inaccurate representation of the object.

Instead, you can use the “|||” flag at the end of the name of the spacing rule to pass special conditions to the DRC engine. In the following example, the “Projecting” command is used to instruct DRC to consider only those edges that project onto each other by more than one manufacturing grid.

**Standard DRC rule:**

- **Rule name:** Min. Permalloy Width
- **Type:** Min Width
- **Rule distance:** 650um
- **Layer1:** Permalloy
Equivalent Calibre format rule:

"Min Permalloy width" { @ < 650 Microns
INTERNAL Permalloy < 650 SINGULAR
SAVE "Min Permalloy width::Result"
}

Standard DRC rule with ||| flag:

Rule name: Min Permalloy width ||| PROJ > 5
Type: Min Width
Rule distance: 650um
Layer1: Permalloy

Equivalent Calibre format rule with ||| flag:

"Min Permalloy width" { @ < 650 Microns PROJ > 5
INTERNAL Permalloy < 650 PROJECTING SINGULAR
SAVE "Min Permalloy width::Result"
}
Specifying DRC Standard Design Rules

Select **Tools > DRC Setup** from the menu, highlight **DRC Standard Rule Set** in the rule set list, then press the Edit button ( ) to open the **Setup DRC Standard Rule Set** dialog, which allows you to modify the DRC standard design rules.

**Rule set**

- **Name** identifies the design rule set.

- **All-angle tolerance** is a value $T$, common to all rules in the set, which together with the distance $D$ for each rule (see below) determines the precision of error checking. A distance on the layout must be less than $D - T$ to be flagged as a violation. The All-angle tolerance is only applied if one of the two edges being compared is non-orthogonal.

- **Limit number of errors reported per rule to** is an integer $T$, with a default value of 1000.
### Rules list

The list of available rules. The checkbox next to each rule indicates if the rule is currently enabled. Enabled rules will be checked when DRC is run.

You can alter the **Rules list** with the following options:

- **Enable All**—Enables all valid rules in the **Rules list**.
- **Disable All**—Disables all rules in the **Rules list**.
- **Add**—Adds a new rule to the **Rules list**. To add a rule, click *Add*, then type the name of the new rule in the **Rule** field.
- **Delete**—Deletes the highlighted rule.
- **Copy**—Adds a copy of the highlighted rule to the **Rules list**. The copy is placed underneath the original rule, with “Copy of” preceding the rule name.
- **Move Rule**—Click the up or down **Move Rule** arrows to reposition the highlighted rule.

#### Rule

The name of the rule highlighted in the **Rules list**. **Rule type**, **Ignore**, **Rule distance**, and **Rule layers** all pertain to the uniquely named rule.

#### Rule type

Selected by clicking the appropriate option button. See “Design Rule Types” on page 535 for information on the supported types.

#### Ignore

Cases which will not be considered a design rule violation. Options include:

- **Coincidences**
- **Intersections**
- **If layer 1 completely outside layer2**
- **Acute angles**

See “Rule Exceptions” on page 538 and “Acute Angles” on page 539 for further information on the use of these options.

#### Rule distance

The distance value associated with a rule. Distances are measured either in display units. You can change the display units using the pull-down menu in the locator bar.

#### Rule layers

DRC specifies which layers are involved in each of the design rules. For example, selecting the **Spacing** rule type automatically specifies **Minimum Layer [ ] to Layer [ ] spacing**. To specify a rule layer, open the **Setup Layers** dialog, then choose from the layer list.

#### Geometry Flags...

Opens the **Geometry Flags** dialog for flagging instances of specific geometry configurations. (See “Geometry Flags” on page 543.) Geometry flags are counted as errors during a DRC run.

#### Export to Command File

Exports the DRC ruleset to a textual command file. The ruleset is opened in a text window, which can then be saved to disk.

---

**Geometry Flags**

During DRC, each drawn layer is checked for geometry violations. Hidden layers are not checked. Use the **Geometry Flags** dialog to specify geometry configurations that L-Edit will flag as errors during DRC, to enable geometry checks on derived layers or on layers that are not used by the active DRC rule.
set. To access this dialog, click the **Geometry Flags** button at the bottom of the **Setup Design Rules** dialog. You can also open this dialog using **Tools > Check Geometry Flags**.

**Flag acute angles**

Reports an error whenever two consecutive edges of a polygon or wire on a drawn layer form an acute angle (less than 90 degrees). These errors are reported in a rule named “Polygons and wires with acute angles (<90°).”

**Flag all-angle edges**

Reports an error when an all-angle edge (i.e., neither orthogonal nor 45-degrees) is found in a polygon on a drawn layer. All-angle edges are measured in cell coordinates; an edge is flagged as all-angle if it is neither orthogonal nor 45-degrees as it occurs in the cell in which it was drawn.

Because angles are measured in cell coordinates, instance rotations do not affect which edges are flagged as all-angle:

Errors are reported to a rule named “All-Angle edges.”
### Flag off-grid
Reports an error when an off-grid vertex or instance is found. Wires are checked using their centerline vertices. Errors are reported to a rule called “Offgrid objects.”

The number in parentheses gives the gridsize, in display units, used to determine offgrid objects and vertices. The gridsize used is the **Manufacturing Grid** size, which you can specify in **Setup > Application—Grid**.

### Flag zero-width wires
Reports an error whenever a wire with zero width is found on a drawn layer.

### Flag polygons with more than \(N_1\) vertices
Reports an error whenever a polygon with more than the specified number of vertices is found on a drawn layer.

### Flag wires with more than \(N_2\) vertices
Report an error whenever a wire with more than the specified number of vertices is found on a drawn layer.

### Flag self intersections (Always checked)
Reports an error whenever a self intersecting polygon or wire is found on a drawn layer. The perimeter boundary of the wire is checked and reported for self intersections, not the centerline. Self intersecting polygons and wires are not processed further by the DRC engine.

### Flag round join/end wire styles (Always checked)
Reports an error whenever a wire with round join/end style is found on a drawn layer. Wires with the round join/end style are processed by the DRC engine, but are approximated as layout style.

### Optimizing Performance
Design rule checking is a complicated, computation-intensive process that involves large numbers of comparisons and measurements. Some DRC runs can result in very long execution times. This section outlines several ways to achieve faster results.

#### Checking Incrementally
Use region-only checks at convenient stopping points in the design process. Performing region-only checks will also help prevent compounding of errors which might require extensive layout modification to correct. Use the full-cell check for completed errors and at least once on the final design.

#### Checking Hidden Layers
You can set whether DRC will or will not check rules involving hidden layers, for each layer that is hidden. For example, if you have examined and repaired all violations involving particular layers, you may want to hide these layers to reduce execution times for subsequent DRC runs involving other layers.
When DRC is initiated, L-Edit will prompt you for your preferred action for each hidden layer it finds.

Yes
Keep this layer hidden and ignore it during DRC.

Yes to All
Keep all layers hidden and ignore them during DRC.

No
Cancel operation.

Show All Layers and Start
Change state of all hidden layers to “shown” and include them in DRC operations.

Disabling Rules

Use the checkboxes in **Setup Design Rules** to enable (checked) or disable (unchecked) rules you do not need. Alternately you can use the ||| flag to append Calibre rules to —see “Flag to Append Special Commands” on page 540.
Design rules, in their simplest form, are usually minimum allowable values for *widths, separations, extensions*, and *overlaps* of and between geometrical objects. The exact nature of design rules is dependent on specifications supplied by the foundry to which the design will be submitted for fabrication.

To check a layout for design rule violations involves two basic steps:

- Define the rules that are acceptable for your design.
- Run the design rule checker, on the entire design or a portion of it.

The commands **Tools > DRC** (for whole cells) and **Tools > DRC Box** (for limited regions) run a *design rule checker*, which determines whether a design obeys a specific set of rules. Design rule violations are saved in the TDB file. You can then step through and display design rule violations using the **Verification Error Navigator**. A summary report of design rule violations is also saved in the TDB file.

Note that the netlist will not be exported if a design rule error exists.

### Design Rule Sets

Design rules are supported in these formats:

- Tanner DRC Standard Rule Set— Tanner format, with graphical setup interface.
- Mentor Graphics Calibre® compatible format — text format command file.*
- Cadence Dracula® compatible format — text format command file.*
- Cadence Assura® compatible format — text format command file.*

### Setting Up DRC

To run DRC you must first load and select the rule sets you want to run. You add and select rule sets from the **Setup DRC** dialog. The Tanner DRC Standard Ruleset is loaded by default. To use it, simply click on its checkbox.

You also use this dialog to set the order in which rule checks are executed, and as a shortcut to open the windows where rule sets can be edited.

Note that checkmarks in the **Setup DRC** list control only which rule sets to run. All other functions in the setup dialog are performed on the rule set that is highlighted.

---

* Calibre is a trademark of Mentor Graphics Corporation. Dracula and Assura are trademarks of Cadence Design Systems, Inc.
Use **Tools > DRC Setup** or press the **Setup DRC** icon ( ) in the verification toolbar to open **Setup DRC**.

**DRC Rule Sets to run**

Lists the DRC command files that are loaded and available to run. **DRC Standard Rule Set** is the built in Tanner rule set which is loaded by default. It cannot be deleted from the list.

DRC will only run the rule sets with checkboxes in the “checked” state.

See “Icons for the DRC Rule Sets list” on page 550 for a table of the icons that control the rule list.

**Calibre Defines**

Use this field to enter variable or variable and value combinations to trigger preprocessor commands that are written in the rule files using **#DEFINE** and **#IFDEF**.

**Note:** DRC setup values in this field are saved and will not replace Extract define values.

**Setup Assura**

See “Assura Setup” on page 826.
Run in background

Check this box to run DRC in background. When DRC is run in background, you can continue to edit and perform other L-Edit operations while DRC is running. DRC results are returned as soon as they are found, so you can browse and correct errors before the entire DRC job is complete.

Lock participating cells while DRC runs

Locks the cell, and all hierarchy below that cell, to prevent edits while DRC is running.

Save node-highlighting data

A check in this box allows you to save node highlighting data to the location specified using the To Folder button (which opens a standard Browse for Folder window).

Note: The Setup Extract—Options dialog and the Setup > Design—Node Highlighting dialog both include the option to save node highlighting data, and to choose where it is saved. Be careful to manage these settings to avoid losing or overwriting your data.

Send E-mail to:

Enter an E-mail address to send notification to the specified recipient when the Extract job is complete. Use Configure to set the E-mail options shown below. Note that most E-mail applications will require a response prior to sending an E-mail initiated from another application.

Configure Mail Service

[Image of Configure Mail Service window]

Play sound

Check to play a sound when DRC is complete. You can configure the sound from the standard Windows sounds available.

Pop up message box

Opens a DRC Completion Report when DRC is complete.
Running DRC

When your layout is complete, you should check for design rule violations before sending the layout to the chip foundry for fabrication. If a chip is fabricated with design rule violations, it may fail to function as designed.

You can run DRC against an entire cell or a specific region (“DRC box”) of the cell.

**Design Rule Check on a Full Cell**

To perform a rule check on the entire layout of the active cell, select **Tools > DRC** ( ).

This button is a toggle, so that during the design check it functions as a stop button ( ). If you stop a DRC run, L-Edit prompts for a confirmation and opens a DRC completion report indicating that the job was terminated prior to completion.

**Region-Only Design Rule Check**

To perform a rule check on just an area of a cell, use **Tools > DRC Box** ( ).

Run a region-only check when a restricted area or group of objects in the layout needs to be checked for design rule errors. A region-only check is useful for interim verification during layout creation or to confirm that a design rule violation in a specific region has been corrected.

To perform a region-only check, select **Tools > DRC Box** then click and drag in the layout (you will be using the DRAW mouse button) to outline the rectangular area of the layout to be checked. The outline will only be visible while DRC is running.

Objects inside or intersecting the region are processed in the **DRC Box** check. Objects intersecting the region are not clipped to the region. As a result, violations can be reported when running region-only
DRC that are not present when DRC is run on the entire cell, due to the exclusion of objects outside the region. If this happens, making the region larger should reduce these false errors.

**Single Rule Check from a Command File**

You can run a single rule in a command file by placing the cursor on the desired rule in the command file, right-clicking the mouse, and selecting **Run Rule rulename**. This will run the selected rule on the entire cell.

After you invoke **Run Rule**, L-Edit opens a dialog where you can choose the file and cell on which to run the specified rule.
DRC Progress

While DRC is running, L-Edit displays a progress dialog similar to the following:

The progress dialog displays the cell, the rule set, and the name of the current rule being checked, as well as the elapsed time, the estimated time remaining in the DRC run, the number of flags checked and rules completed, and the number of errors found.

For each rule for which errors are found, the dialog lists the rule name and the corresponding number of violations. The contents of the progress dialog are saved in the DRC Summary Report. The DRC Summary Report may be opened at any time from the Verification Error Navigator, by invoking Actions > Open DRC Summary Report.

Notification of DRC Completion

DRC can be configured to notify you upon completion with either a brief summary report, a sound, or both. This is especially useful when running DRC in background. To configure DRC to present a
message box upon completion, select **Pop up message box** in the **Setup DRC** dialog. The dialog shown below will be presented when the DRC job is complete.

![DRC Completion Report](image)

- **Activate view of this cell** Opens the cell that the DRC job was run on. This is useful when running DRC in background, and a different cell may currently be open.
- **Show Summary Report** Displays the DRC summary report in a text window. See summary reports, below.

To configure DRC to play a sound upon completion, select **Play sound** in the **Setup DRC** dialog. Press the **Configure** button to invoke the standard Windows sound selection dialog.

## Command File Syntax Checking

L-Edit DRC includes a syntax checker for checking the validity of command files. The syntax checker should be used to verify the correctness of command files before running DRC and also when you create new command files.

The layout and `.ext` file must be open when L-Edit performs this check. To invoke the syntax checker, select **Tools > Check Syntax** (shortcut key **F6**).

- A window will open below the command file window listing syntax errors and warnings. Double-clicking the cursor on a line in the error window will scroll the command file window to the corresponding error.
• Syntax errors within connect() and/or device() statements are reported with the line number and a best guess of the reason for parser failure. Semantic errors are reported as they are for Calibre, Assura or Dracula rule decks.

• Bookmarks will also be placed in the command file window on the line corresponding to each syntax error. You can use Edit > Go To Next Bookmark (shortcut key F2) and Edit > Go To Previous Bookmark (shortcut key Shift+F2) to navigate through the errors. Edit > Clear All Bookmarks (shortcut key Ctrl+Shift+F2) clears all bookmarks.

**DRC Status**

Each cell has a DRC Status setting, which can be one of the following states:

• **Needed** — DRC has never been run on the cell, or edits have been made since last run.

• **Passed** — DRC has been run on the cell, and no violations were found.

• **Failed** — DRC has been run on the cell, and violations were found.

Running DRC on a cell will set the DRC Status flag for the cell to either **Passed** or **Failed**. The DRC status flag is set only for the toplevel cell from which the DRC job is invoked, even though instances of other cells were checked in the process.

Any change to the DRC Setup, including changes to Layer Setup, will cause the DRC Status of all cells to revert to the **Needed** state.

However, because non-edit operations such as changes to the lock status, version number or default view mark a cell as changed, L-Edit no longer sets the DRC status to **Needed** whenever a cell is changed. (This behavior is new as of version 16.00. Prior to that release, editing a cell after a DRC run would revert it’s status to **Needed**.)

The DRC status of all cells may be viewed in the Design Navigator (See “DRC Status” on page 247), or for any single cell it may be viewed in the **Cell > Info** dialog (See “Listing the Object Types or Layers Used in a Cell” on page 241). The DRC Status may also be manually changed in the **Cell > Info** dialog.

**Excluding Cells from DRC**

Instances of specified cells can be excluded from DRC. To mark a cell for exclusion from DRC, select **Exclude instances of this cell from DRC** in the **Cell > Info** dialog of the cell.

Marking a cell for exclusion from DRC is particularly useful for logo cells, which typically contain DRC violations that can be ignored.

**Debugging DRC Results with Generated Layers**

When debugging DRC results, it is often useful to be able to generate and visualize the derived layers that are used in a rulecheck command.
**Generating Layers**

You can generate any of the derived layers within the global scope of a command file with a single command, **Tools > Generate Layers** (see “Generating Derived Layers” (page 458) for more details). Global scope consists of any definition not within the braces (`,{`) of a rulecheck statement.

**Generate Layers directly from a Command File**

You can also generate a single layer in a command file by placing the cursor on the desired layer definition in the command file, right-clicking the mouse, and selecting **Layer Derivation** in the context sensitive menu. Layers defined in either global and local scope of a rulecheck statement can be selected.
When you execute **Layer Derivation** from a command file, L-Edit opens a dialog that displays the derivation tree for the layer, and allows you to generate that layer as well as intermediate layers in the derivation. You also choose the file and cell for which to generate layers.

Pressing the **Generate** button will generate the selected layer. Intermediate layers in the derivation can be included by checking the **Generate intermediate layers** checkbox.
This section provides the reference to Calibre® compatible DRC command file format.

## Input Conventions

This section describes the syntax and constraints on commands and functions.

### Case Sensitivity

Keywords are always case-insensitive. Names are case-insensitive unless used for cell names or file names. RuleCheck names, layer names, net names, variable names, and so forth, are always case-insensitive.

### New Line Insensitivity

The statements that appear in a rule can begin anywhere on a line and can span lines. In addition, statements and operations need not each begin on a new line.

### Navigation in the Text Editor

To speed navigation in the text editor, you can right-click on a layer name to access the command “Go To Layer Definition” in the context-sensitive menu, which jumps the cursor to the line with the definition for that layer.

### Preprocessor Commands

#DEFINE, #IFDEF, #ELSE, and #ENDIF are supported as a mechanism of conditionally executing blocks of commands.

### Comments

Commands may be commented out using C-Style (/ * ... */) and C++ Style (//) comments characters. C-Style comments may span multiple lines. C++ Style comments extend from the comment characters to the end of the line.

### Constraints

Many operations require a mathematical constraint as one or more of the input parameters. The constraint is usually applied to either a count of the number of some quantity, or to the measurement of some distance. Constraints are expressed as follows:
< a  
> a  
<= a  
>= a  
== a  
!= a  
> a < b  
>= a < b  
> a <= b  
>= a <= b

Where "a" and "b" are non-negative numbers. Not all operations permit all constraints.

**Numeric Expressions**

A numeric expression can be used to specify any numeric parameter in any layer operation. Numeric expressions can also be used to define variables in a Variable specification statement. A numeric expression is a combination of numeric constants, numeric variables, the unary "+" and "-" operators, and the binary "+", ",", "*", "/", and "%" operators.

**Reserved Symbols**

The following are reserved symbols and may not be used in layer names or rule names in the command file.

```
// @ = { } " " ( ) [ ] < == > <= != . + * / ! % && || :: /* */
```
Reserved Keywords

The following words are reserved keywords and may not be used as layer names or rule names in the command file.

<table>
<thead>
<tr>
<th>abut</th>
<th>factor</th>
<th>parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>acute</td>
<td>flag</td>
<td>perimeter</td>
</tr>
<tr>
<td>and</td>
<td>flatten</td>
<td>perp</td>
</tr>
<tr>
<td>angle</td>
<td>group</td>
<td>perpendicular</td>
</tr>
<tr>
<td>angled</td>
<td>grow</td>
<td>polygon</td>
</tr>
<tr>
<td>area</td>
<td>holes</td>
<td>precision</td>
</tr>
<tr>
<td>by</td>
<td>in</td>
<td>proj</td>
</tr>
<tr>
<td>coin</td>
<td>include</td>
<td>projecting</td>
</tr>
<tr>
<td>coincident</td>
<td>inside</td>
<td>rectangle</td>
</tr>
<tr>
<td>convex</td>
<td>int</td>
<td>rectangles</td>
</tr>
<tr>
<td>copy</td>
<td>interact</td>
<td>region</td>
</tr>
<tr>
<td>corner</td>
<td>internal</td>
<td>resolution</td>
</tr>
<tr>
<td>cut</td>
<td>intersecting</td>
<td>shrink</td>
</tr>
<tr>
<td>donut</td>
<td>layer</td>
<td>singular</td>
</tr>
<tr>
<td>drawn</td>
<td>length</td>
<td>size</td>
</tr>
<tr>
<td>drc</td>
<td>measure</td>
<td>snap</td>
</tr>
<tr>
<td>enc</td>
<td>merge</td>
<td>square</td>
</tr>
<tr>
<td>enclose</td>
<td>not</td>
<td>step</td>
</tr>
<tr>
<td>enclosure</td>
<td>obtuse</td>
<td>title</td>
</tr>
<tr>
<td>exclude</td>
<td>offgrid</td>
<td>touch</td>
</tr>
<tr>
<td>expand</td>
<td>opposite</td>
<td>variable</td>
</tr>
<tr>
<td>ext</td>
<td>or</td>
<td>vertex</td>
</tr>
<tr>
<td>extend</td>
<td>out</td>
<td>with</td>
</tr>
<tr>
<td>extent</td>
<td>outside</td>
<td>xor</td>
</tr>
<tr>
<td>extents</td>
<td>overlap</td>
<td></td>
</tr>
<tr>
<td>external</td>
<td>para</td>
<td></td>
</tr>
</tbody>
</table>

**Note:**
Note that as a workaround you can enclose a keyword in double quotation marks, which will cause it to be treated as a string constant and highlighted in red.
Command File Examples

The smallest command file consists of:

- A PRECISION statement to define the ratio of user units to database units
- LAYER statement(s) to define drawn layers
- A RuleCheck statement

Note that the LAYER statement can be used to give a layer a different name in the command file compared to the tdb file. One could have layer Metal1 with GDS number 49 in the tdb file, but define LAYER M1 49 in the command file.

```
PRECISION 1000
LAYER M1 49
M1.S { @ M1 Spacing < 0.3
       EXT M1 < 0.3
}
```

A Basic Command File

```
//***************************************************************
//  Sample DRC Command File
//***************************************************************
TITLE "Sample DRC Command File"

// Setup Info
PRECISION 1000
RESOLUTION 10
FLAG ACUTE YES
FLAG SKEW YES
FLAG OFFGRID YES
FLAG NONSIMPLE YES

// Input Layers
LAYER ACTIVE 1
LAYER POLY 2
LAYER CONTACT 3
LAYER METAL1 4

// Common Derived Layers
FIELD_POLY = POLY NOT ACTIVE
POLY_CONT = CONTACT NOT OUTSIDE FIELD_POLY
DIFF_CONT = CONTACT OUTSIDE FIELD_POLY

// POLY DRC Rules
PO.1 { @ Minimum poly space < 0.38
       EXT POLY < 0.38 ABUT < 90
   }
PO.2 { @ Minimum poly on field space to active < 0.16
       EXT POLY ACTIVE < 0.16 ABUT < 90 SINGULAR
   }
PO.3 { @ Minimum active extend gate < 0.6
```
ENC POLY ACTIVE < 0.6 SINGULAR ABUT < 90

// CONTACT DRC Rules
CO.1 { @ Contact width != 0.30
         NOT RECTANGLE CONTACT == 0.30 BY == 0.30
}
CO.2 { @ Contact spacing < 0.30
         EXT CONTACT < 0.30 SINGULAR
}
CO.3 { @ POLY contact space to active < 0.28
         EXT POLY_CONT ACTIVE < 0.28 SINGULAR ABUT <90
}
CO.4 { @ Active overlap contact < 0.12, also floating contacts
         ENC DIFF_CONT ACTIVE < 0.12 SINGULAR ABUT <90 OUTSIDE ALSO
}

// METAL1 DRC Rules
M1.1 { @ METAL1 width < 0.36
         INT METAL1 < 0.36 SINGULAR ABUT < 90
}
M1.2 { @ METAL1 spacing < 0.36
         EXT METAL1 < 0.36 ABUT < 90
}
M1.3 { @ Min METAL1 density < 30%
         DENSITY METAL1 < 0.30
}

Command Usage

Commands can be used as layer derivation statements, or in RuleCheck statements. Layer derivations that appear outside of Rule Check statements are referred to as “global” derivations. Layer derivations inside of Rule Check statements are referred to as “local” derivations.

- A layer derivation statement consists of directing the results of a command to a named layer.
  
  GATE = POLY AND ACTIVE

- An implicit layer definition consists of a matched pair of parenthesis containing a layer definition.

  NTRAN = (POLY AND ACTIVE) NOT NWELL

A RuleCheck statement consists of a name followed by a left brace “{”, followed by a sequence consisting of either layer derivations or dimensional checks, followed by a right brace “}”:

- A Rule Check Statement directs the results of a command to the Verification Error Navigator, with the specified rule name.

  rule-name1 {
    EXT GATE < 2.0
  }

Rule check statements have the following properties:

- Multiple commands within a Rule Check Statement are allowed.

  rule1 {
    EXT GATE < 2.0
Layer derivations such as “Z = Layer1 AND Layer2” are allowed within a rule check statement. Layer Z is local in scope within that rule, and can be used by commands within that rule.

```plaintext
rule2 {
    GATE = POLY AND ACTIVE
    INT GATE < 2.0
}
```

Any command, not only dimensional check operations, may be used to direct errors to the Verification Error Navigator. For example

```plaintext
rule3 {
    VIA OUTSIDE METAL
}
```

Rule Check comments are text following the @ symbol, to the end of the line. Multiple Rule Check comment line are allowed. Rule check comments are displayed in the Verification Error Navigator along with the rule name.

```plaintext
rule4 { @ Rule check comment
    @ Second comment line
    GATE = POLY AND ACTIVE
    INT GATE < 2.0
}
```

Implicit layer definitions are allowed within dimensional check operations (ENC, EXT, INT). Implicit layer definitions may not be used with dimensional check operations when the edge directed output options [ ] or ( ) are used.

```plaintext
rule5 {
    EXT (POLY AND ACTIVE) < 2.0
}
```

**Intermediate Layer Rules**

**Edge Directed Output**

The output of a dimensional check operation (ENC, EXT, INT) can be put on an edge layer by surrounding one of the input layers in the operation with brackets [ ] or parenthesis ( ). The edge layer may then be processed by other commands.

Enclosing a layer in brackets is called positive edge-directed output, and returns the edges on the layer that are normally flagged by the rule.

Enclosing a layer in parenthesis is called negative edge-directed output, and returns the edges on the layer that would not normally be returned by the rule.
Only one edge-directed output specification may appear in a single dimensional check operation. Edge-directed output specifications apply to Internal, External, and Enclosure.

```
rule-name {
    Z = EXT [layer1] < n1
    LENGTH Z > n2
}
```

```
rule-name {
    Z = EXT [Layer1] Layer2 < n1
    LENGTH Z < n2
}
```

**Polygon Directed Output**

The output of a dimensional check operation (ENC, EXT, INT) can also be put on a polygon layer by using the REGION option. The polygon layer may then be processed by other commands.

```
rule-name {
    Z = INT Metal1 < n1 REGION
    EXT Z > n2
}
```

**Output Reports**

L-Edit produces a summary report showing all rules that were run and all rules that were turned off, and the rules enabled or disabled by the DRC SELECT CHECK and DRC UNSELECT CHECK options.

**Summary and Classification of Commands**

Commands can be classified as either layer selectors or layer constructors. Commands classified as layer selectors select existing polygon or edge data from the appropriate input layer. Commands classified as layer constructors create new polygon data.

**Polygon Layer Selectors**

The table below lists the polygon layer selector operations. These commands select polygons from an input layer.

- [Not] Area
- Copy
- [Not] Cut
- [Not] Donut
- [Not] Enclose
- [Not] Enclose Rectangle
- [Not] Inside
- [Not] Interact
- [Not] Net
<table>
<thead>
<tr>
<th>Edge Layer Selectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Not] Outside Perimeter</td>
</tr>
<tr>
<td>[Not] Rectangle</td>
</tr>
<tr>
<td>[Not] Touch Vertex</td>
</tr>
<tr>
<td>[Not] With Edge</td>
</tr>
<tr>
<td>[Not] With Neighbor</td>
</tr>
<tr>
<td>[Not] With Text</td>
</tr>
<tr>
<td>[Not] With Width</td>
</tr>
</tbody>
</table>

The table below lists the edge layer selector operations. These commands select edges from an input layer.

- [Not] Angle
- [Not] Coincident Edge
- [Not] Coincident Inside Edge
- [Not] Coincident Outside Edge
- Convex Edge
- Copy
- Drawn Acute*
- Drawn Offgrid *
- Drawn Skew*
- Enclosure (edge-directed dimensional check)
- External (edge-directed dimensional check)
- [Not] Inside Edge
- Internal (edge-directed dimensional check)
- [Not] Length
- Offgrid*
- [Not] Outside Edge
- Path Length
- [Not] Touch Edge
- [Not] Touch Inside Edge
- [Not] Touch Outside Edge
Layer Constructors

Operations classified as layer constructors create new polygon data. The table below lists the layer constructor operations.

AND
[Not] Inside Cell
Density
Enclosure
(polygon-directed check)
Expand Edge
Expand Text
Extent
Extents
External (polygon-directed check)
Flatten
Grow
Holes
Internal (polygon-directed check)
Merge
NOT
OR
Pathchk
Rectangles
Shrink
Size
Snap
Stamp
XOR

Omitted Environment Setup Commands

The following environment setup commands are not supported as they are not required when running DRC in the L-Edit environment.

DRC RESULTS DATABASE It is not necessary to specify an external results filename. Results are saved in the tdb file.
DRC CHECK MAP
This statement controls the database output structure in Calibre. It is not required in HiPer Verify.

DRC CHECK TEXT
This statement controls what components of a rule in the command file get copied into the Calibre results database. It is not required in HiPer Verify.

DRC EXCLUDE FALSE NOTCH
This statement overrides the default Calibre behavior to suppress certain false errors. It is not required in HiPer Verify.

DRC MAXIMUM VERTEX
A user specified limit on the number of vertices on error polygons is not supported. Polygon fracturing on layout is supported by the Draw > Convert > Fracture Polygons command.

DRC RESULTS DATABASE
It is not necessary to specify an external results filename. Results are saved in the tdb file.

DRC KEEP EMPTY
Specifies whether rule checks containing no errors are saved to DRC results. HiPer always saves all rule checks so one can see the rule was checked, even when there are no errors. Checks with no errors can be made visible or not visible in the Verification Error Navigator.

DRC SUMMARY REPORT
It is not necessary to specify an external summary file name. A summary report is always created and saved in the tdb file. It may be opened in a text window from the Verification Error Navigator toolbar with Actions > Open DRC Summary Report, and then saved to disk.

LAYOUT DEPTH
Checking toplevel polygons only is not supported. The entire hierarchy is processed.

LAYOUT PATH
LAYOUT PATH is not required in HiPer Verify. DRC is run on the currently active cell in the open tdb file.

LAYOUT PRIMARY
DRC is run on the currently active cell. LAYOUT PRIMARY is not required in HiPer Verify.

LAYOUT SYSTEM
In Calibre, specifies the layout file type. Not required in HiPer Verify.

The indicated options are not supported in the following commands.

AND
Single layer syntax is not supported. AND layer1 [constraint]

EXTENT CELL
EXTENT CELL is not supported.
Function Overview

Functions may be grouped in the following categories.

Environment Setup

- **TITLE** (page 695)
- **PRECISION** (page 679)
- **RESOLUTION** (page 685)

Operating Commands

Operating commands control high level aspects of the DRC job.

- **DRC UNSELECT CHECK** (page 609)
- **DRC UNSELECT CHECK** (page 609)
- **DRC PRINT AREA** (page 605)
- **DRC PRINT PERIMETER** (page 606)
- **DRC SELECT CHECK** (page 607)
- **DRC TOLERANCE FACTOR** (page 608)
- **DRC UNSELECT CHECK** (page 609)
- **GROUP** (page 639)
- **INCLUDE** (page 642)
- **SVRF ERROR** (page 692)
- **VERTEX** (page 701)

Hierarchy Modification Commands

These commands modify cell hierarchy for all layers in the cell:

- **EXCLUDE CELL** (page 617)
- **EXPAND CELL** (page 618)
- **FLATTEN CELL** (page 637)
- **FLATTEN INSIDE CELL** (page 638)

These commands modify the hierarchy of specified layers, across all cells:

- **MERGE** (page 657)

Geometry Flags

Geometry flags are checked on original drawn polygons and wires only. The check is performed on the original polygons and wires on the layer, not on a merged representation of the layer. Only drawn layers are checked, derived layers are not checked.

- **LAYER RESOLUTION** (page 655)
Chapter 20: HiPer Verify: Calibre Command Files

- FLAG NONSIMPLE (page 630)
- FLAG OFFGRID (page 631)
- FLAG POLYGONVERTEXLIMIT (page 632)
- FLAG SKEW (page 633)
- FLAG WIREVERTEXLIMIT (page 634)
- FLAG ZEROWIDTHWIRES (page 635)
- DRAWN ACUTE (page 600)
- DRAWN OFFGRID (page 601)
- DRAWN SKEW (page 602)
- LAYER RESOLUTION (page 655)
- OFFGRID (page 670)

Drawn Layer Definitions

- POLYGON (page 677)
- LAYER MAP (page 654)

Connect and Connectivity Commands

A NET is formed by the CONNECT command, as shown below:

```
LAYER layer1 49
LAYER layer2 51
LAYER connect-layer 50
CONNECT layer1 layer2 BY connect-layer
```

The LAYER command will read both layout and text from the specified layer; no extra command is required to read in text. Nets may be named by placing text labels (L-Edit ports) overlapping the polygons that form a net, thus assigning the net a name equal to the text of the label. For text labels that are boxes or lines, the center of the text label is used when comparing overlaps with polygons.

Database Specification Commands for Net Naming

The TEXT LAYER and TEXT DEPTH commands specify the layers and the hierarchical levels from which text labels are used for net naming, as follows:

- TEXT LAYER - Only text objects on layers that are listed in the TEXT LAYER command will be used for net naming.
- TEXT DEPTH - The TEXT DEPTH command specifies the depth in the hierarchy for using text objects for application in net naming. TEXT DEPTH ALL uses text objects from throughout the hierarchy, TEXT DEPTH PRIMARY uses only text objects from the top-level cell, and TEXT DEPTH n1 uses text objects n1 levels below the top-level cell. TEXT DEPTH PRIMARY is the default. Text objects that come from lower levels of the hierarchy are flattened and transformed to the top-level coordinate space. These text objects then behave as if they originated at the top level.

Priority Rules for Attachment of Net Names

The process of assigning label names to nets proceeds in the following order:
Explicit Attachment - ATTACH

If the rule file contains the command

```
ATTACH text-layer geometry-layer
```

connectivity extraction will assign the name of a text object on layer text-layer to a net containing a polygon on layer geometry-layer if the polygon on layer geometry-layer completely covers the text object. The rule file can contain more than one Attach operation for the text layer, such as:

```
ATTACH text-layer geometry-layer1
ATTACH text-layer geometry-layer2
...
ATTACH text-layer geometry-layerN
```

In this case, the connectivity extractor looks for polygons on any one of the target layers geometry-layer1, ..., geometry-layerN that intersect the label location. If exactly one polygon is found, then the label name is assigned to the net that contains that polygon. If more than one polygon is found, one is chosen arbitrarily and a warning is issued.

Implicit Attachment - Label and polygon on the same layer

The connectivity extractor will assign the name of a text object on a layer to a net containing a polygon on the same layer if the polygon completely covers the text object.

Attachment by LABEL ORDER

After net naming by explicit Attach commands or Implicit Attach has taken place, the order of layers specified by the LABEL ORDER command is used to resolve any additional net naming attachments. The connectivity extractor will assign the name of a text object on any layers listed in the TEXT LAYER command that have not already been explicitly or implicitly attached, to the first overlapping polygon in the LABEL ORDER list.

If no Label Order operation is present in the rule file, or if no polygon on any of the Label Order layers intersects the label location, then the label is ignored and a warning is issued (Unattached Label Warning).
Antenna Rules

Antenna checks are used to limit the damage of the thin gate oxide during the manufacturing process due to charge accumulation on the interconnect layers (metal, polysilicon) during certain fabrication steps like Plasma etching.

The term *antenna* refers to large metal interconnect, connected to a gate that is not electrically connected to silicon or grounded, during the processing steps of the wafer. If the connection to silicon does not exist, charges may build up on the interconnect to the point that rapid discharge takes place and permanent physical damage results to thin transistor gate oxide. This rapid and destructive phenomenon is known as the "antenna effect" or "Plasma Induced Damage".

The *antenna ratio* is defined as the ratio between the physical area of the conductors making up the antenna to the total gate oxide area to which the antenna is electrically connected. The area of the conductor includes both the top of the conductor as well as the side walls, although one of these might be ignored in some checks.

- NET AREA (page 659)
- NET AREA RATIO (page 660)
- NET AREA RATIO PRINT (page 668)
- ORNET (page 672)
- POLYNET (page 678)

Polygon Boolean Operations

- AND (page 575)
- OR (page 671)
- NOT (page 669)
- XOR (page 709)

Utility Layer Generation Operations

- EXENT (page 627)
- EXTENTS (page 628)
- RECTANGLES (page 684)
- SNAP (page 690)

Polygon Size Operations

- HOLES (page 641)
- SHRINK (page 688)
- SIZE (page 689)
- WITH WIDTH (page 708)

Two-Layer Polygon Selection Operations

These commands select polygons from an input layer based on relationships to polygons on another input layer.
Single-Layer Polygon Selection Operations

These commands select polygons from a single input layer, based on the properties of the polygons on that layer.

- DONUT (page 599)
- ENCLOSE RECTANGLE (page 616)
- RECTANGLE (page 680)
- VERTEX (page 701)

Polygon Area Operations

- AREA (page 577)
- DENSITY (page 587)

Polygon Edge Operations

- WITH EDGE (page 705)
- EXPAND EDGE (page 619)

Edge, Length and Angle Operations

- ANGLE (page 576)
- LENGTH (page 656)
- PATH LENGTH (page 675)

Edge Selection Operations

- TOUCH EDGE (page 697)
- COINCIDENT INSIDE EDGE (page 580)
- COINCIDENT OUTSIDE EDGE (page 581)
- INSIDE EDGE (page 645)
- TOUCH EDGE (page 697)
- TOUCH INSIDE EDGE (page 698)
- TOUCH OUTSIDE EDGE (page 699)
Dimensional Check Operations

Dimensional edge operations (ENC, EXT, INT) measure the distance between pairs of edges of polygons on different or the same layer, and output results if the specified distance constraint is met. To determine the output of the measurement of the distance between a pair of edges, a region is constructed around each edge that extends out from the edge a distance specified by the constraint amount, and with a shape specified by the measurement metric. Portions of the opposing edge that intersect the region of a given edge are then output.

The region construction is illustrated below for the three available measurement metrics. The region construction is shown for one of the edges of layer1, and corresponding output from edges on layer2 is shown. Region construction is similarly performed on layer2, and output edges from layer1 is determined. Note that by default, perpendicular edges are not measured.

**Euclidean** — The Euclidean metric forms a region with quarter-circle boundaries at that extend past the corners of the selected edges. This is the default metric.

```
rule_euc {EXT layer1 layer2 < 2}
```

![Euclidean Region Construction](image1)

**Square** — The Square metric forms a region with right-angle boundaries that extend past the corners of the selected edges.

```
rule_sq { EXT layer1 layer2 < 2 SQUARE }.
```

![Square Region Construction](image2)

**Opposite** — The opposite metric forms a region with right-angle boundaries that do not extend past the corners of the selected edges.

```
rule_op { EXT layer1 layer2 < 2 OPPOSITE }.
```

![Opposite Region Construction](image3)
rule_opp { EXT layer1 layer2 < 2 OPPOSITE}.

- ENC (page 610)
- EXT (page 622)
- INT (page 646)

Text Based Operations

- EXPAND TEXT (page 621)
- WITH TEXT (page 707)

Netlist Extraction Operations

- DEVICE (page 590)
Functions
**AND**

\[
\text{[polygon-layer =]} \text{ layer1 AND layer2}
\]
\[
\text{[polygon-layer =]} \text{ AND layer1 layer2}
\]

**Description**

Calculates the intersection of layer1 and layer2.

**Parameters**

- **layer1**
  - A drawn or derived polygon layer
- **layer2**
  - A drawn or derived polygon layer
ANGLE

\[\text{edge-layer} = \text{NOT}\] \text{ANGLE layer1 constraint}\n
**Description**

Produces all layer1 edges whose acute angle magnitude with the x-axis conforms to the constraint. The NOT option produces all layer1 edges not produced by the corresponding ANGLE operation.

**Parameters**

| **layer1** | A drawn or derived polygon layer. |
| **constraint** | Constraint is any range defined by “Constraints” (page 491) where the constraint's values represent an angle range in degrees. The constraint values must be greater than or equal to 0 but less than or equal to 90. |
AREA

[polygon-layer =] [NOT] AREA layer1 constraint

**Description**

Produces layer1 polygons whose area conforms to the constraint. The NOT option produces layer1 polygons not produced by the corresponding AREA operation.

**Parameters**

- **layer1**: A drawn or derived polygon layer
- **constraint**: Constraint is any range defined by “Constraints” (page 491) where the constraints values must be a non-negative real number.
ATTACH

ATTACH layer1 layer2

Description

The attach operation assigns names to extracted nets using text objects (ports) placed on the layout. It attaches the name of a text object on layer1 to a net containing a polygon on layer2 if the polygon completely covers the text object. The same layer1 can be used in multiple ATTACH operations with different layer2 names.

Parameters

layer1          A drawn layer name containing net labels.
layer2          A drawn layer name or layer set or a derived polygon layer. Must appear as an input layer to a Connect or Sconnect operation.
COINCIDENT EDGE

[edge-layer =] [NOT] COINcident EDGE layer1 layer2
[edge-layer =] layer1 [NOT] COINcident EDGE layer2

Description

Produces all layer1 edges or edge segments coincident with layer2 edges. The NOT option produces all layer1 edges not produced by the corresponding COINCIDENT EDGE operation.

Parameters

layer1   A drawn or derived polygon layer or a derived edge layer.
layer2   A drawn or derived polygon layer or a derived edge layer.
COINCIDENT INSIDE EDGE

\[
\begin{align*}
\text{[edge-layer =]} & \ [\text{NOT}] \ \text{COINCident INside EDGE} \ \text{layer1 layer2} \\
\text{[edge-layer =]} & \ \text{layer1} \ [\text{NOT}] \ \text{COINCident INside EDGE} \ \text{layer2}
\end{align*}
\]

**Description**

Produces all layer1 edges or edge segments coincident from the inside with layer2 edges. An edge segment on layer1 is coincident from the inside with an edge segment on layer2 if the two edges are coincident and interior of the two polygons corresponding to the two edges are overlapping in the region adjacent to the two edges. The NOT option produces all layer1 edges not produced by the corresponding COINCIDENT INSIDE EDGE operation.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived polygon layer or a derived edge layer.</td>
</tr>
<tr>
<td>layer2</td>
<td>A drawn or derived polygon layer or a derived edge layer.</td>
</tr>
</tbody>
</table>
**COINCIDENT OUTSIDE EDGE**

\[
\text{[edge-layer =]} \ \text{[NOT]} \ \text{COINcident OUTside EDGE} \ \text{layer1} \ \text{layer2} \\
\text{[edge-layer =]} \ \text{layer1} \ \text{[NOT]} \ \text{COINcident OUTside EDGE} \ \text{layer2}
\]

**Description**

Produces all layer1 edges or edge segments coincident from the outside with layer2 edges. An edge segment on layer1 is coincident from the outside with an edge segment on layer2 if the two edges are coincident and the polygon on layer1 is outside the polygons on layer2 in the region adjacent to the two edges. The NOT option produces all layer1 edges not produced by the corresponding COINCIDENT OUTSIDE EDGE operation.

**Parameters**

- **layer1**: A drawn or derived polygon layer or a derived edge layer.
- **layer2**: A drawn or derived polygon layer or a derived edge layer.
CONNECT

Syntax 1:
CONNECT layer1 [...layerN]

Syntax 2:
CONNECT layer1 layer2 [...layerN] BY layerC

Description

Syntax 1 forms a connection between abutting or overlapping objects on the input layers. Connected objects are part of the same electrical node. Syntax 2 specifies electrical connections between layer2, layer1, and layerC objects, where layer2 (through layerN) objects have positive area overlap with both layer1 and layerC objects.

Parameters

layer1[... layerN]  A drawn or derived layer, followed optionally by other drawn or derived layers.

layer2[... layerN]  A drawn or derived layer, followed optionally by other drawn or derived layers.

BY layerC  A required keyword for Syntax 2, followed by a drawn or derived layer. This specifies a contact layer.

Description Details

The layers in the connect operation without the BY keyword may appear in any order without changing the meaning of the operation.

A connect operation with the BY keyword specifies a connection between layer1, layerC, and one of the layer2 through layerN objects. The connection is made to the first of the layer2 through layerN objects, that has a positive area overlap with both layer1 and layerC objects.
CONVEX EDGE

Simple syntax:

[edge-layer =] CONVEX EDGE layer1 endpoint_constraint [WITH LENGTH edge_length_constraint]

Detailed syntax:

[edge-layer =] CONVEX EDGE layer1
ANGLE1 angle_constraint [LENGTH1 abut_length_constraint]
ANGLE2 angle_constraint [LENGTH2 abut_length_constraint]
[WITH LENGTH edge_length_constraint]

Description

Produces an edge layer by selecting edges depending on the number of convex endpoints. Edge length, angle of abutting edges, and length of abutting edges may also be considered. This operation is not limited to selection of edges based purely on convexity. In most cases the convexity (or concavity) of the endpoints of edges is a by-product of the properties you desire. This operation has two syntaxes, simple or detailed. Both the simple and detailed specifications can include the WITH LENGTH secondary keyword and the edge_length_constraint. The differences between simple and detailed are discussed in the following sections.

Simple Syntax Parameters

layer1 A drawn or derived polygon layer or a derived edge layer.

Detailed Endpoint Specification

Edge selection may also be specified by the angles formed by adjacent edges at each endpoint of a given edge. ANGLE1 denotes one endpoint and ANGLE2 denotes the other endpoint. LENGTH1 and LENGTH2 additionally specify edge selection based on the length of the adjacent edges at the endpoints of a given edge. Length parameters are associated with ANGLE1 and ANGLE2, respectively.

Angle constraints which are < 180 degrees (measured internal to the polygon) are convex. Angle constraints which are > 180 degrees are concave. If you do not specify LENGTH1 and LENGTH2 they default to the constraint >=0.

Detailed endpoint specification selects edges with the following algorithm.

- Designate the endpoints of the edge as A and B.
- The edge is selected if the status at endpoint A satisfies the ANGLE1 and LENGTH1 constraints, and the status at endpoint B satisfies the ANGLE2 and LENGTH2 constraints.
- If the edge is not selected from the previous test, then:
  - The edge is selected if the status at endpoint B satisfies the ANGLE1 and LENGTH1 constraints and the status at endpoint A satisfies the ANGLE2 and LENGTH2 constraints.

The tool does not select the edge if it does not meet the above requirements.
Edge Layer Input

The process of edge selection needs to be refined for a derived edge input layer. This is due to the possibility that abutting edges may not be present. For simple endpoint specification, the number of convex endpoints is counted as before. However, if an abutting edge is absent, the endpoint is not counted. This edge is then selected if the number of convex endpoints passes the associated constraint. For detailed endpoint specification, the angle at an endpoint where there is no abutting edge is defined as 0 degrees. This value can be used since edges cannot meet at an angle of 0 degrees in a layer representing merged data. The length of the abutting edge (which is missing) to be is defined to be 0 user units. For example, if the ANGLE1 constraint includes 0, an endpoint with no abutting edge will satisfy the ANGLE1 constraint.

Examples

The following example selects all metal1 edges which have both endpoints at a convex corner:

   CONVEX EDGE metal1 == 2

The following example selects all metal edges which have one endpoint at a convex corner and length less than 3 user units:

   CONVEX EDGE metal == 1 WITH LENGTH < 3
COPY

[result-layer =] COPY [layer1]

Description

Copies a polygon or edge layer to a new layer with a different name. Can also be used to output a layer as DRC violations.

Parameters

layer1 A drawn or derived polygon layer or a derived edge layer.

Examples

The COPY command can be used to output a derived layer as errors:

GATE = POLY AND ACTIVE
Gate.Out { // Rule to output GATE derived layer
  COPY GATE
}

The COPY command can be used to output intermediate layers in a Rule Check statement as errors for visualization purposes.

LongThinMetalTraces { @ Flag Metal traces narrower than 4.0 um
  @ and longer than 100.0 um
  A = INT [Metal1] < 4.0
  COPY A // COPY command outputs all A edges as errors
  LENGTH A > 100.0 // Errors will also be output from here.
  // Comment LENGTH out if desired.
}
CUT

[polygon-layer =] layer1 [NOT] CUT layer2 [constraint [BY NET]]
[polygon-layer =] [NOT] CUT layer1 layer2 [constraint [BY NET]]

Description

Produces layer1 polygons that have portions both inside and outside layer2. The NOT option produces those layer1 polygons not produced by the corresponding CUT operation. The input layers will be merged if the constraint option is used, and will run slower than the CUT operation with no constraint.

(This option is not yet supported.) If BY NET is specified, the constraint applies to the number of layer2 polygons on distinct nets that CUT layer1.

Parameters

- layer1: A drawn or derived polygon layer
- layer2: A drawn or derived polygon layer
- constraint: Specifies the number of layer2 polygons that layer1 must have portions both inside and outside of, in order to be selected by the CUT operation. Constraint is any range defined by “Constraints” (page 491) where the constraints values must be positive integers.
- BY NET: When the BY NET option is used, then a layer1 polygon is selected when the specified number of layer2 polygons on distinct nets cut the layer1 polygon. Layer2 must have connectivity.

Examples

```
layer1 layer2 layer1 CUT layer2
```

[Diagram showing layer1 and layer2 with an example of CUT operation]
**DENSITY**

```
[polygon-layer =] DENSITY layer1 [...layerN] [[density_expression]]
constraint [INSIDE OF { {x1 y1 x2 y2} | EXTENT | (LAYER layer2) } ]
[WINDOW {w | wx wy} ] [STEP {s | sx sy} ] [TRUNCATE | BACKUP|IGNORE|WRAP]
[CENTERS value] [{PRINT | PRINT ONLY} file_name]
```

**Description**

Produces rectangular boundaries within which the ratio of the area of layer1 to the area of the rectangle meets the constraint. The boundaries within which density is calculated may be a single rectangle, or using the WINDOW/STEP options, a sequence of rectangular windows moving across the chip can be considered. The INSIDE OF LAYER option modifies the boundary within which density is calculated from a rectangle to being the boundaries of the polygons on the specified layer.

**Options Not Supported**

By Polygon, Centered, Corner, Gradient, RDB

**Parameters**

- `layer1 [...]layerN` A drawn or derived polygon layer
- `density_expression` An optional expression in brackets ( [ ] ) that allows customizable control over the density ratio computations. The expression may contain numbers, numeric variables, binary operators ( *, /, +, -), unary operators (+, -, !, ~), and AREA functions of the input layers of the form:
  ```
  AREA (input_layer)
  ```
  The expression must not result in negative values.
- `constraint` Constraint is any range defined by “Constraints” (page 491) where the constraints values must be a non-negative real number. Specifies the ratio of the area of layer1 to the area of the specified boundary that must exist for the boundary polygon to be produced.
- `INSIDE OF` Defines a rectangular boundary within which a moving window will travel. If this option is not specified, then the boundary is equal to the database extent.
  - `INSIDE OF X1 Y1 X2 Y2` Specifies a rectangular boundary. Negative numbers must be in parenthesis.
  - `INSIDE OF EXTENT` Specifies that the boundary is the Extent of the input layer.
  - `INSIDE OF LAYER layer2` Specifies that the density is calculated within the boundary of each polygon on layer2.
- `WINDOW` Specifies a window within which the density check is computed.
  - `WINDOW w` Specifies a square window with dimension w.
  - `WINDOW wx wy` Specifies a rectangular window with dimension wx by wy.
- `STEP s` Specifies that window moves up and to the right by a distance s.
Density is computed as a ratio of two areas:

\[
    \text{Density} = \frac{A_1}{A_2}
\]

**If the WINDOW option is not present then:**

If no INSIDE OF options are present, then A1 is the area of the input layer, layer1, and A2 is the area of the database extent.

If INSIDE OF EXTENT is specified, then A1 is the area of the input layer, layer1, and A2 is the area of the extent of layer1.

If INSIDE OF x1 y1 x2 y2 is present, then let R be the rectangle formed by x1 y1 x2 y2. A1 is the area of (layer1 AND R ) and A2 is the area of R.

If INSIDE OF LAYER layer2 is specified, then for each polygon P on layer2, the density is calculated using A1 = area of (layer1 AND P), and A2 = area of P, and the boundary of P is output if the constraint is met.

**Description Details**

**STEP sx sy**

Specifies that window moves to the right by sx and up by sy.

**TRUNCATE**

Specifies that if the rightmost or topmost windows do not exactly end at the boundary of the database (or layer), then those windows are truncated to fit the database (or layer), and the density calculation is done on the reduced window size. This is the default behavior.

**BACKUP**

Specifies that if the rightmost or topmost windows do not exactly end at the boundary of the database (or layer), then those windows are moved such that the rightmost or topmost boundary of the window coincides with the boundary of the database (or layer), and the density calculation is done on a full window size.

**IGNORE**

Specifies that if a window overlaps the right-hand edge or the top edge of the boundary box, the window is ignored and no data for that window location is output.

**WRAP**

Specifies that if a window overlaps the right-hand edge or the top edge of the boundary box, the boundary box and its data are duplicated and added to the right-hand side or top side of the original bounding box. The density measurement is then taken in the window that intersects the duplicated boundaries.

**CENTERS value**

Specifies that output should be squares of dimension value, located at the center of each window that would be normally be output.

**{PRINT/PRINT ONLY} file_name**

PRINT and PRINT ONLY writes the coordinates and corresponding density of output rectangles to the specified file. The PRINT option prints to the file and outputs to the resulting polygon layer or error layer. The PRINT ONLY option prints to the filename but does not output to the result layer or error layer.

A path that includes the colon character must be in quotes. If a full path is not provided, then the path is relative to the tdb file.
If the WINDOW option is present:

If no INSIDE OF options are present, then a rectangle, W, specified by the WINDOW dimensions is positioned starting at the lower left of the database extent, and subsequently positioned by moving by the STEP amount in the x- and y- directions. For each window position, A1 is calculated as the area of (layer1 AND W) and A2 is the area of W.

If INSIDE OF EXTENT is specified, then a rectangle, W, specified by the WINDOW dimensions is positioned starting at the lower left of layer1 extent, and subsequently positioned by moving by the STEP amount in the x- and y- directions. For each window position, A1 is calculated as the area of (layer1 AND W) and A2 is the area of W.

If INSIDE OF x1 y1 x2 y2 is present, then let R be the rectangle formed by x1 y1 x2 y2. A rectangle, W, specified by the WINDOW dimensions is positioned starting at the lower left of R, and subsequently positioned by moving by the STEP amount in the x- and y- directions. A1 is the area of (layer1 AND R AND W) and A2 is the area of the rectangle formed R AND W.

If INSIDE OF LAYER layer2 is specified then a rectangle, W, specified by the WINDOW dimensions is positioned starting at the lower left of the extent of layer2, and subsequently positioned by moving by the STEP amount in the x- and y- directions. For each window position, A1 is calculated as the area of (layer1 AND layer2 AND W) and A2 is the area of W.

Examples

1. Density of POLY over the database extent must exceed 15%.

   POLY_Density { @ Min. POLY area coverage 15%
                     DENSITY POLY < 0.15 }

2. Density of Metal1 in every 200um x 200um window must exceed 25%. Windows are stepped at 100um interval, and rightmost and topmost windows are backed up. Print violating windows to a log file.

   METAL1_Density { @ METAL1 area coverage must be >= 25% over 200 um x 200 um
                    DENSITY METAL1 < 0.25 WINDOW 200 STEP 100 BACKUP PRINT METAL1_density.log }

3. Density of METAL1 plus METAL2 must be greater than 40%

   METAL_Density { @ METAL1 plus METAL2 area coverage must be >= 40% over 200umx200um
                   DENSITY METAL1 METAL1 < 0.4 WINDOW 200 STEP 100 BACKUP PRINT METAL_density.log
                   [ (AREA(METAL1) + AREA(METAL2))/AREA() ]
                 }
DEVICE

DEVict {element_name [(model_name)]} device_layer {pin_layer [(pin_name)]} 
[<auxiliary_layer>][BY NET | BY SHAPE][property_specification][NETLIST MODEL
 netlist_model_name] [NETLIST ELEMENT netlist_element_name]

Description

The DEVict statement defines devices for netlist extraction, defines how devices are to be recognized, names pins, and defines properties of devices and how they are computed.

Parameters

- **element_name**
  The type of the device, which may be either built-in or user-defined. Predefined element names specify built-in (or reserved) devices. All other devices are user-defined (or generic).

- **model_name**
  An optional string, enclosed in parentheses, that specifies the model for the device. It is also known as the component subtype of the device. If specified, it must immediately follow the element_name parameter.

- **device_layer**
  A required layer containing the device recognition shapes. It is also called the seed layer. The device layer is the first layer you specify within the operation given the recognition precedence of the element_name as discussed previously. Device recognition is centered around each device (or seed) shape on this layer. You can specify the device_layer as one of the pin_layers, but not more than once per statement. Connectivity need not be established on these layers.
**pin_layer**

A required layer on which pin shapes for the device are found. You can specify multiple *pin_layer* names. These layers must have their connectivity established by connectivity extraction operations. A layer has connectivity when one of the following conditions is satisfied:

- A Connect operation contains the layer.
- A net-preserving operation derives the layer from a separate layer carrying a net number
- A Stamp operation transfers net numbers to the layer.

Device recognition logic, not the connectivity extractor, associates pin shapes to device shapes. Therefore, you do not need to connect the pin layers to the device layer by Connect statements. To do so would short the pins together through the device shape. Pins using the same layer in a Device statement are interchangeable and belong to the same pin-swap group.

Note that pin shapes must touch or overlap device recognition shapes in order for a device to be recognized. Touching at a corner point does not satisfy this condition. Device shapes that do not touch the correct pin layers are considered *bad devices*, and are listed in the Verification Error Navigator.

If the *element_name* in the statement is a reserved name, then you must specify one layer for each pin. If a device has more than one pin on a given layer, you must repeat that layer as a parameter for each associated pin. The same device, auxiliary, and pin layers can appear in multiple Device statements. However, to prevent ambiguity, the following restriction is enforced: if two statements have the same *device_layer*, then it must not be possible to reorder the list of auxiliary and pin layers in one statement so that it exactly matches the list of auxiliary and pin layers in the other statement. In making the comparison, the *device_layer* should be ignored in any list where it appears as a *pin_layer*. For example:

```
device foo A B(p) A(q)
device bar A B(p)
```

causes a compilation error since pin A in foo is ignored. This results in a device ambiguity the compiler does not allow.

**Note:** If the device pin number does not match the default pin number of that device type, then the device should be extracted as a subckt instead of the defined device type.

For example, the following command defines device type R (resistor) and 3 pin layers (pd3bbTerm_S, pd3bbTerm_S, ntubNotRes).

```
DEVICE R("pprnwb") pd3bbResBody pd3bbTerm_S pd3bbTerm_S ntubNotRes
```

We know the resistor should only have two pins. So the pin layer number does not match the default resistor device pin number. In this case, this device should be extracted as “X” device (subckt) instead of “R” device.
(pin_name)

An optional name, enclosed in parentheses, that explicitly names each pin in the definition of a device. The parameter pair `pin_layer (pin_name)` can be specified any number of times in one statement.

If the `element_name` in the statement is built-in for recognition, then default `pin_names` are assigned. You can specify the `device_layer` as one of the pin layers, but not more than once per statement. Pin names do not need to be explicitly provided, but if present, they must match the default names that would have been assigned if you want to use the built-in algorithms for property calculation. Providing pin names for built-in-for-recognition elements makes the rule file more readable and is recommended.

If the element has extra pins in addition to the default pins, then a `pin_name` parameter must be provided following each extra `pin_layer` in the Device statement. The order of the extra pin layers is unimportant in the Device statement. For the extra pins, you can create any pin names you wish, except reserved keywords must be enclosed in double quotation marks.

If the `element_name` in the statement is user-defined, then a `pin_name` parameter must be provided following each `pin_layer` in the Device operation. The `pin_layer` order is unimportant in the Device operation in this case. However, the order of pin layers can determine the order in which pins are listed in an extracted netlist. You may use any pin names as desired, except if you use a reserved keyword for a `pin_name` then you must enclose the `pin_name` in double quotes.

The same `pin_name` parameter cannot be used twice in the same Device statement, but can be reused in other statements. If the schematic for the device uses certain pin names, then the same names should be used in the Device operation so that LVS applications can match correct pins.
**<auxiliary_layer>**

An optional layer name, enclosed in angle brackets ( < > ), that identifies non-pin layers used to classify device instances and used in property computation. You can specify `<auxiliary_layer>` any number of times in one statement. They can appear before, after, or intermixed with the pin layers.

An auxiliary layer is a layer containing shapes that are neither seed nor pin shapes. You can use auxiliary layers to interact with other shapes of the device for property alteration, or to classify the device. For each auxiliary layer present in a Device operation, at least one shape from that layer must touch or overlap the seed shape before a device instance can be extracted. You cannot limit the number of overlapped shapes, although a property computation can determine the number of shapes found to touch or overlap by using the COUNT( ) function of the built-in property computation language.

Auxiliary layers do not need to have connectivity. Any node information on auxiliary layers is unavailable to device extraction. An auxiliary layer cannot appear twice in the same Device statement. You cannot use a layer as both an `<auxiliary_layer>` and a *pin_layer* in the same Device operation, nor in any other Device operation using the same *device_layer*.

**BY NET | BY SHAPE**

An optional keyword set that selects the pin recognition method for the device. Two device recognition operations in the rule file with the same *device_layer* must have the same pin recognition method. The possible choices are:

**BY NET** — Treats pin shapes on the same layer and connected to the same net as a single pin. This is the default behavior if you do not include a keyword from this set. The pin fill-in algorithm is used to supply missing pins, where applicable.

**BY SHAPE** — Treats pin shapes as separate pins even when they are on the same layer and are connected to the same net. Also, no fill-in is attempted to supply missing pins. Specifying BY SHAPE can increase runtime significantly.
[**property_specification**] An optional string, enclosed in square brackets ([ ]), that specifies which properties are to be computed for device instances and how they are computed.

A property specification can take one of two forms: either a list of floating-point numbers, or a short program written in a property computation language. The list of numbers form is only applicable to certain reserved element names.

The following is an example of each form of property specification:

```
DEVice R resistor_layer metal_1 metal_2 [1.1]
// resistivity is 1.1 resistance units per square
// internal default calculation is used to find
// resistance
```

```
DEVice R layer metal_1 metal_2

[ property R
  R = .5*(AREA(layer)-AREA_COMMON(layer, metal_1)-
  AREA_COMMON(layer, metal_2))
]
/* R is defined as the resistance of the device and
calculated as shown */
```

If you provide a property specification algorithm, you take on responsibility for calculating all properties of the device. The device is then considered user-defined, even if it has a reserved element name.

[**NETLIST MODEL**

`netlist_model_name`] An optional keyword set that specifies to use `netlist_model_name` when netlisting element model names in SPICE, otherwise, the `(model_name)` parameter is used. These are only used for netlisting and not as component subtypes during LVS. For example:

```
DEVICE mn ngate poly(g) ndiff(s) ndiff(d) psub(b)
NETLIST MODEL nmos
```

causes the device to appear in the netlist as a model named `nmos`.

[**NETLIST ELEMENT**

`netlist_element_name`] This optional keyword set overrides the default SPICE element name prefixes during netlisting. The default names are: C, D, J, M, Q, R, and X (X is the netlist name for user-defined devices). NETLIST ELEMENT is not used by LVS for circuit comparison and is not used for extracted netlists. In the examples below, LDD devices are netlisted as M devices, and L devices are netlisted as y devices:

```
DEVICE LDD gate sd(D) poly(G) sd(S) bulk (B) NETLIST ELEMENT "m"
```

```
DEVICE L seed contact(POS) contact(NEG) NETLIST ELEMENT "y"
```

In addition, a character can be specified to separate the instance name from the device ID number. In the example below, X devices are netlisted as x$n where $n is the device ID number:

```
DEVICE X seed a(A) b(B) NETLIST ELEMENT "x$"
```

This example causes the device to appear in the netlist as the type `netlist_element_name`. 
DISCONNECT

Parameters

There are no parameters for this statement.

Description

Allows the total deletion of an existing connectivity model in an Incremental Connect sequence. (Incremental Connect sequences occur when Connect operations are performed incrementally due to the DRC Incremental Connect YES specification statement in the rule file.) You can specify this statement any number of times.

See Also

“NET” (page 545), “CONNECT” (page 541), “SCONNECT” (page 546) and “DRC INCREMENTAL CONNECT” (page 543).
DMACRO and CMACRO

Description

Macros are functional templates, similar to the macros of the C and C++ languages, that can be called multiple times in a rule file.

Defining Macros—DMACRO

A macro definition consists of the keyword DMACRO (define macro), a name, zero or more arguments, followed by "{" bracketing a sequence of zero or more verification statements or operations, closed by "}".

DMACRO names must be unique, each argument must be a name, and an argument may not be duplicated in the same DMACRO argument list.

Ordinary nesting rules for left and right braces prohibit nested DMACRO definitions.

Invoking Macros—CMACRO

A macro is invoked by the keyword CMACRO (call macro), followed by a macro name and a list of zero or more arguments. Each argument may be either a name or a numeric constant. The name must match that of some DMACRO definition, and a sufficient number of arguments must be present after the CMACRO name. The DMACRO definition between the braces is placed in the CMACRO call, with argument substitution.

When a DMACRO is instantiated in a CMACRO, the layer definition names defined in the DMACRO are locally-scoped. You do this by generating a name that is unique with in the rule file for each layer definition in the DMACRO construct when the DMACRO is instantiated. References to layer definition names in the DMACRO are also substituted appropriately, as shown in the shown width checks example.

A DMACRO definition may itself contain nested CMACROS. However, recursion is not allowed, so no CMACRO can call a DMACRO that contains a call to that same CMACRO.

Rule Files

You can use macros to derive layers; however, a DMACRO definition used to derive layers, either locally or globally, may not have any intermediate derived layers. For example, this is allowed:

DMACRO ex1 layer constraint {
(size layer BY constraint UNDEROVER) and layer
}

x = CMACRO ex1 metal1 .01 // global
rule_1 {
  x = CMACRO ex1 metal1 .01 // local
  copy x
}

Notice that DMACRO ex1 has no intermediate derived layers in it, so deriving x, as shown, will work in either local or global scope. However, the following example demonstrates a problematic syntax:

DMACRO ex2 layer1 layer2 constraint{
int_layer = layer1 or layer2
// intermediate derived layer
size int_layer BY constraint UNDEROVER
}
x = CMACRO ex2 in1 in2 .01
// Bad. intermediate layer exists in ex2

Notice DMACRO ex2 has an intermediate derived layer in it. This means ex2 should not be used in a layer derivation, because the scoping of intermediate derived layers in a DMACRO cannot generally be used to derive other layers.

Any of the rule file comment characters may be used in a macro definition; however, user comments will not appear in the DRC results database. User comments should be placed in the rule check that calls the DMACRO.

**Examples**

```
PRECISION    1000
RESOLUTION    100

LAYER Poly   46
LAYER Active 43
LAYER Metal1 49
LAYER Metal2 51
LAYER EmptyLayer 100

// Simple Macro
DMACRO WideLayout layer value {
    (SIZE layer BY value UNDEROVER) and layer
}

WidePoly {
    CMACRO WideLayout Poly 1.5
}

// Local scoping
// Example showing that layer definitions are locally scoped.
// Layers A and B are scoped locally within the macro
// so there is no confusion between the two CMACRO calls
// in the same rulecheck.

DMACRO WidthCheck layer size_val wid_val {
    A = SIZE layer BY -size_val
    B = SIZE A BY size_val
    INT B < wid_val
}

widthChecks {
    CMACRO WidthCheck Poly 1.0 5.0
    CMACRO WidthCheck Metal1 1.0 5.5
}

// Nested Macros
DMACRO NarrowLayer layer size_val {
    INT layer <= size_val
}

DMACRO NarrowLayout layer1 layer2 size_value {
    CMACRO NarrowLayer layer1 size_value
    CMACRO NarrowLayer layer2 size_value
```
Narrows {
    CMACRO NarrowLayout Metall Poly 1.0
}
DONUT

[polygon-layer =] [NOT] DONUT layer1 [constraint]

Description

Produces all layer1 polygons that have a hole or holes. The NOT option produces those polygons not produced by the corresponding DONUT operation.

Parameters

layer1 A drawn or derived polygon layer
constraint Specifies the number of holes a layer1 polygon must have in order to be selected by the DONUT operation. Constraint is any range defined by “Constraints” (page 491) where the constraints values must be integer.

Examples

Show a polygon with a hole that touches the boundary at a point - this isn't really a hole.
**DRAWN ACUTE**

**Description**

Reports an error for any two consecutive edges of a drawn polygon or wire that form an acute angle.

This command produces the same output as the FLAG ACUTE command, but the drawn command provides for a user specified rule name.

**Examples**

G.1 { @ Shapes with acute angles between line segments are not allowed
DRAWN ACUTE
}

```
DRAWN OFFGRID

**Description**

Reports an error for every offgrid vertex of drawn polygons and wires. Also reports errors for offgrid instance placements, rotated instances, and instance arrays and scaling that could result in offgrid geometry. Checking and reporting is done in the context of the cell. The grid is defined by the RESOLUTION command.

This command produces the same output as the FLAG OFFGRID command, but the drawn command provides for a user specified rule name.

**Examples**

```plaintext
G.2 { @ grid must be an integer multiple of 0.005u
DRAWN OFFGRID
}
```
**DRAWN SKEW**

**Description**

Reports an error for any non-90 or non-45 degree edge of a drawn polygon or centerline segment of a drawn wire. Reporting is based on the angle of the edge in its cell coordinate space, not in coordinate space of instances of that cell.

This command produces that same output as the FLAG SKEW command, but the drawn command provides for a user specified rule name.

**Examples**

```plaintext
G.3 { @ Shapes must be orthogonal or on a 45 degree angle.
       DRAWN SKEW
   }
```
DRC INCREMENTAL CONNECT

Parameters

YES  Enables incremental connectivity.

NO   Use this setting so that incremental connectivity is not enabled. This is the default behavior if you do not include this statement in your rule file.

Description

Used only in DRC applications, in particular for antenna checking and other specialized connectivity checks. Enables incremental connectivity extraction which, if used, causes the tool to connect a subset of the interconnect layers and perform design rule checks based on the connectivity of that subset.

When enabled, L-Edit views the rule file as having the following partial front-to-back ordering so that incremental connect rules are read first:

```
<layer operations>  //Connectivity zone #0
<connect operations>
<layer operations>  //Connectivity zone #1
<connect operations>
<layer operations>  //Connectivity zone #2
...
<connect operations>
<layer operations>  //Connectivity zone #N
```

Layer operations include the output operations of DRC. As such, operations requiring connectivity information in connectivity zone #0 are not allowed. Operations requiring connectivity information in a zone #x where x > 0 treat the connectivity as if only the Connect operations prior to connectivity zone #x have been executed. They are not allowed if that connectivity can only be established by Connect operations after connectivity zone #i.
DRC MAXIMUM RESULTS

DRC MAXIMUM RESULTS \( \{ \text{maxresults} \mid \text{ALL} \} \)

**Description**

Specifies the maximum number of errors that will be reported for a DRC Rule. You can only specify this statement once in a rule file. When the maximum results are generated for a rule, a warning is issued, and no additional results are added to the Verification Error Navigator.

For the Geometry Flags, FLAG ACUTE, FLAG OFFGRID, and FLAG SKEW, a maximum of 100 errors are always reported.

**Parameters**

- **maxresults**
  
  A positive integer that specifies the maximum result count for an individual RuleCheck in DRC execution. If this statement is not included, the default value is 1000.

- **ALL**
  
  A required secondary keyword that specifies that there is no maximum result count, for an individual RuleCheck in DRC.
DRC PRINT AREA

DRC PRINT AREA layer [... layer]

Description

Prints the area of the specified layers to the DRC Summary Report. This statement can appear multiple times.

Parameters

layer

A drawn or derived polygon layer.

Examples

DRC PRINT AREA metal1 metal2
DRC PRINT PERIMETER

DRC PRINT PERIMETER layer [... layer]

Prints the perimeter of the specified layers to the DRC Summary Report. This statement can appear multiple times. This operation requires flattening the layer and can therefore be time consuming.

Parameters

layer

A drawn or derived polygon layer.

Examples

DRC PRINT PERIMETER metal1 metal2
DRC SELECT CHECK

DRC SELECT CHECK rule_check ...

Description

Provides selective inclusion of specified RuleCheck statements or RuleCheck Groups in the DRC job. By default all rules are included. The wildcard (?) is supported for this command.

RuleCheck statements are selected for inclusion as follows:

1. If there are no DRC Select Check specification statements in the rule file then all rules are included. Otherwise, only those RuleChecks specified in DRC Select Check statements are included.

2. All RuleChecks specified in any DRC Unselect Check specification statements in the rule file are then excluded.

Parameters

rule_check A list of RuleCheck or Group names.

Example 1

DRC SELECT CHECK PO.S1 PO.S2 PO.S3

Example 2

GROUP Poly_CHK PO?
PO.1 {
    INT Poly < 2.
}

PO.2a {
    EXT Poly < 3.
}
DRC TOLERANCE FACTOR

DRC TOLERANCE FACTOR tolerance

Description

Reduces false errors on all-angle DRC rule check operations. The tolerance is applied to distance measurement operations whose constraint is of the form “< d” when either one of two edges being compared is non-orthogonal. In this case, the constraint value is decreased by the given tolerance (to no more than 0) prior to the actual measurement. This command can appear only once in the command file.

Parameters

tolerance

A positive real number in user units that specifies the tolerance used in DRC when at least one of the edges is all-angle. Tolerance is assigned a default value of 1/PRECISION.

Examples

DRC TOLERANCE FACTOR 0.005
DRC UNSELECT CHECK

DRC UNSELECT CHECK rule_check ...

Description

Provides selective exclusion of specified RuleCheck statements or RuleCheck Groups in the DRC job. By default all rules are included.

RuleCheck statements are selected for inclusion as follows:

1. If there are no DRC Select Check specification statements in the rule file then all rules are included. Otherwise, only those RuleChecks specified in DRC Select Check statements are included.

2. All RuleChecks specified in any DRC Unselect Check specification statements in the rule file are then excluded.

Parameters

rule_check A list of RuleCheck or Group names.

Examples

DRC UNSELECT CHECK PO.S1 PO.S2 PO.S3
ENC

ENClosure layer1 layer2 constraint [metric] [polygon_containment] [connectivity_filter] [orientation_filter] [projection_filter] [angled_filter] [corner_filter] [intersection_filter] [reversal] [output]

Description

Measures the distance between the outside of layer1 and the inside of layer2 boundaries. Edge pairs that meet the constraint are output. The ENC command is typically used for enclosure or extension checks. Note that the behavior of the rule depends on the order of the layers. By default, intersecting or overlapping edge pairs are not compared.

Parameters

The following parameters are common to all ENC, EXT, and INT RuleChecks.

layer1
A drawn layer, derived polygon layer, or derived edge layer.

layer2
A drawn layer, derived polygon layer, or derived edge layer.

Constraint
Constraint is any range defined by “Constraints” (page 491), except > a, >= a, or != a, where the constraints values must be a non-negative real number in user units.

metric
EUCLIDEAN (Default) — Use the euclidean metric.

OPPOSITE — Use the opposite metric.

SQUARE — Use the square metric.

polygon_containment
MEASURE ALL — Specifies that all edges are compared, including those normally not compared due to intersection, polygon containment, or visibility blocking.

MEASURE COINCident — Specifies that edge pairs that are outside coincident that would normally not be compared, should be compared.

connectivity_filter
CONNECTED — Specifies that only edges from the same net are compared.

NOT CONNECTED — Specifies that only edges from different nets are compared.

To use these filters, the input layers must posses valid connectivity.
orientation_filter

**ACUTE ALSO** — Specifies to measure edges with an appropriate angle between 0 and 90 degrees, exclusive. This is the default behavior if you do not specify a choice from the acute_filter option subset.

**ACUTE ONLY** — Specifies to measure only edges with an appropriate angle between 0 and 90 degrees, exclusive. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same Enclosure operation.

**NOT ACUTE** — Specifies to not measure edges with an appropriate angle between 0 and 90 degrees.

**PARAllel ALSO** — Specifies to measure parallel edges. This is the default behavior if you do not specify a choice from the parallel_filter option subset.

**PARAllel ONLY** — Specifies to measure only parallel edges. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same Enclosure operation.

**NOT PARAllel** — Specifies to not measure parallel edges.

**NOT PERPendicular** — Specifies to not measure perpendicular edges. This is the default behavior if you do not specify a choice from the perpendicular_filter option subset.

**PERPendicular ONLY** — Specifies to measure only perpendicular edges. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same Enclosure operation.

**PERPendicular ALSO** — Specifies to measure perpendicular edges.

**NOT OBTUSE** — Specifies to not measure edges with an appropriate angle between 90 and 180 degrees, exclusive. This is the default behavior if you do not specify a choice from the obtuse_filter option subset.

**OBTUSE ONLY** — Specifies to measure only edges with an appropriate angle between 90 and 180 degrees, exclusive. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same Enclosure operation.

**OBTUSE ALSO** — Specifies to measure edges with an appropriate angle between 90 and 180 degrees, exclusive.

projection_filter

**PROJecting [constraint]** — Specifies to compare the distance between two edges only when one edge projects onto the other edge and the length of projection conforms to the constraint. This is the default behavior, with constraint set to “>=0”, if you do not specify a choice from this set in the operation.

**NOT PROJecting** — Specifies to compare the distance between two edges only when neither edge projects onto the other edge.

angled_filter

**ANGLED [constraint]** — Specifies to measure the two edges only when the number of non-orthogonal (angled) edges in the pair meets the given constraint. The constraint is optional and when omitted, defaults to “> 0”. 
corner_filter

You cannot specify this filter with an orientation, projection, or angled filter.

**CORNER TO CORNER** — Specifies to measure and output errors only for edges in a corner-to-corner configuration.

Additional options **CORNER, CORNER TO EDGE, and NOT CORNER** are not supported at this time.
intersection_filter

Specifies to measure intersecting edges, and specifies exactly the characteristics of intersecting edges that are to be measured. Intersecting or abutting (coincident) edges are not measured by default. Coincident or abutting edges are considered to be intersecting. The value of intersection_filter is:

\[ \text{ABUT [abut_constraint]} \mid \text{OVERLAP} \mid \text{SINGULAR} \mid \text{INTERSECTING ONLY} \]

You can specify any combination of ABUT, OVERLAP, and SINGULAR in one operation.

**ABUT [abut_constraint]** — Specifies that intersecting edges should also be output if the angle between them conforms to the optional constraint (interpreted in degrees). Output from the ABUT condition is in addition to any other output the Enclosure operation generates. The abut_constraint modifier must contain non-negative real numbers less than 180. Single-operator constraints such as \(< 90\) and \(> 135\) are interpreted as \(\geq 0 < 90\) and \(> 135 < 180\). With no constraint specified, the default value is \(\geq 0 < 180\).

If the abut_constraint modifier includes zero in its range then any edges A of layer1 which are coincident inside with edges B of layer2 are also output (because the angle between the exterior side of A and the interior side of B is zero).

**OVERLAP** — Specifies that output should also occur where a polygon from one input layer crosses a polygon from the other input layer. Edges forming the point of overlap are measured and output as if an unconstrained ABUT parameter was specified. This overrides any specified ABUT parameter at the point of overlap only.

This keyword cannot be used when either of the input layers is a derived edge layer because polygons are required to determine if an overlap condition is present. Output from the overlap condition is in addition to any other output generated by the Enclosure operation.

**SINGULAR** — Specifies that intersecting edges at points of polygon singularity should also be output. Singularities are point-to-edge or point-to-point polygon intersections or self-intersections.

**INTERSECTING ONLY** — Specifies that only intersecting edge errors are to be output. Must be preceded with at least one of the other options in this group. The ABUT, OVERLAP, and SINGULAR options normally add intersecting edge violations to the non intersecting edge violations that are normally output. The INTERSECTING ONLY option, when added to one of the other options, causes only intersecting edge violations to be output. This filter ignores orientation, angled, projection, and corner filters.

**Output REGION [EXTENTS]** — constructs polygon projections between the error edges and then outputs them as polygon data. When you specify REGION the polygon region is used, but when you specify REGION EXTENTS the extents of the polygon region is used.
Region Centerline

Constructs derived polygon data similar to REGION, only the output is the centerlines of the polygonal regions, not the regions themselves.

Centerlines are formed prior to the merging of the regions, and are along the direction of the edges whose measurement forms the region. They have a default width of eight database units. The optional parameter value allows you to specify the centerline width. The value must be a floating-point number greater than or equal to two database units.

For two, parallel, horizontal edges, the y-coordinate of the centerline segment is always closer to the bottom edge if any snapping occurs.

The following parameter applies to ENC only.

reversal

INSIDE ALSO — outputs edges from layer2 that are enclosed by the layer1. When you specify INSIDE ALSO between layer1 and layer2, then edges from layer2 inside, but not coincident-inside, relative to layer1 are output by the operation. Layer1 cannot be a derived edge layer.

The connectivity keywords CONNECTED and NOT CONNECTED do affect the output from the INSIDE ALSO option. If you specify NOT CONNECTED, then the INSIDE ALSO parameter does not output an edge if the polygon containing it (or the edge coincident inside with it) are on the same electrical node. If you specify CONNECTED, then the INSIDE ALSO parameter does not output an edge if the polygon containing it (or the edge coincident inside with it) are on different electrical nodes.

OUTSIDE ALSO — outputs edges from layer1 that are not enclosed by layer2. When you specify OUTSIDE ALSO between layer1 and layer2, then edges from layer1 outside or coincident-outside with layer2 are output by the operation. If layer2 is a derived edge layer, then the semantics are restricted to coincident outside edges from layer1.

The connectivity keywords CONNECTED and NOT CONNECTED do not affect output from the OUTSIDE ALSO keyword in an Enclosure operation.

Examples

PO.0.1 { @ Minimum POLY extension out of ACTIVE < 0.25
ENC ACTIVE POLY < 0.25 SINGULAR ABUT >=0 <90
}

VIA1.E.1 { @ Min M1 surrounding VIA1 is 0.1
ENC VIA1 M1 < 0.1 ABUT<90 SINGULAR OVERLAP OUTSIDE ALSO
}
ENCLOSE

[polygon-layer =] layer1 [NOT] ENCLOSE layer2 [constraint [BY NET]]
[polygon-layer =] [NOT] ENCLOSE layer1 layer2 [constraint [BY NET]]

Description

Produces layer1 polygons that completely enclose any layer2 polygon. The NOT option produces those layer1 polygons not produced by the corresponding ENCLOSE operation. The input layers will be merged if the constraint option is used, and will run slower than the ENCLOSE operation with no constraint.

(This option is not yet supported.) If BY NET is specified, the constraint applies to the number of layer2 polygons on distinct nets that CUT layer1.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived polygon layer</td>
</tr>
<tr>
<td>layer2</td>
<td>A drawn or derived polygon layer</td>
</tr>
<tr>
<td>constraint</td>
<td>Specifies the number of layer2 polygons that a layer1 polygon must enclose, in order to be selected by the ENCLOSE operation. Constraint is any range defined by “Constraints” (page 491) where the constraints values must be integer.</td>
</tr>
<tr>
<td>BY NET</td>
<td>When the BY NET option is used, then a layer1 polygon is selected when the specified number of layer2 polygons on distinct nets enclose the layer1 polygon. Layer2 must have connectivity.</td>
</tr>
</tbody>
</table>
ENCLOSE RECTANGLE

[polygon-layer =] ENCLOSE RECTANGLE layer1 width length [ORTHOGONAL ONLY]

Description

Selects all polygons on layer1 that can enclose a rectangle of the specified width and length dimensions. Width may be oriented along either the x- or y- axis, but only orthogonal rectangles or rectangles at 45-degrees will be considered. To consider only orthogonally oriented rectangles, use the ORTHOGONAL ONLY option.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived polygon layer.</td>
</tr>
<tr>
<td>width</td>
<td>A positive floating-point number interpreted in user units.</td>
</tr>
<tr>
<td>length</td>
<td>A positive floating-point number interpreted in user units.</td>
</tr>
<tr>
<td>ORTHOGONAL ONLY</td>
<td>Limits selection to polygons enclosing rectangles having sides that are parallel to the x- and y- axis.</td>
</tr>
</tbody>
</table>
**EXCLUDE CELL**

`EXCLUDE CELL name ...`

**Description**

Excludes all instances of specified cells when processing DRC commands. Cells may also be excluded from DRC processing by checking “Exclude instances of this cell from DRC” in the **Cell > Info** dialog in L-Edit.

This operation automatically finds or matches cells that come from T-Cell and X-Ref cells.

**Parameters**

- **name**

  The name of a cell. If a cell name has spaces, then the name is in quotes. Name can be specified any number of times in one statement. The cell name parameters can contain one or more asterisk (*) wildcard characters, where the (*) matches zero or more characters. When using the * character, enclose the cell name in quotes. Otherwise, a compilation error occurs because the asterisk is a reserved symbol.

**Examples**

`EXCLUDE CELL logo`
EXPAND CELL

EXPAND CELL name ...

Description

Expands all placements of the specified cells one level into the hierarchy of their parents. Typically this statement is used to improve performance in gate arrays by expanding base cell containers down to the level of the base.

This operation automatically finds or matches cells that come from T-Cell and X-Ref cells.

Text is not expanded by this statement and this statement can be specified any number of times.

Parameters

name

The name of a cell. If a cell name has spaces, then the name is in quotes. The cell name parameters can contain one or more asterisk (*) wildcard characters, where the (*) matches zero or more characters. When using the * character, enclose the cell name in quotes. Otherwise, a compilation error occurs because the asterisk is a reserved symbol.

Examples

EXPAND CELL cell32 cell 34
EXPAND EDGE

[polygon-layer =] EXPAND EDGE layer1 expansion_set number1 [EXTEND BY
[FACTOR] number2] [CORNER FILL]

Description

Converts all layer1 edges into rectangles by sweeping edges a specified distance, in a direction toward
the inside of the polygon, or toward the outside of the polygon from which the edge originated.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived polygon layer, or a derived edge layer.</td>
</tr>
<tr>
<td>expansion_set</td>
<td></td>
</tr>
<tr>
<td>INside BY</td>
<td>Expands an edge towards the inside of the polygon it originated from by the amount specified in the number1 parameter. This secondary keyword can be used along with OUTside BY.</td>
</tr>
<tr>
<td>INside BY FACTOR</td>
<td>Expands an edge towards the inside of the polygon it originated from by the product of the number1 parameter and the edge's length. This secondary keyword can be used along with OUTside BY.</td>
</tr>
<tr>
<td>OUTside BY</td>
<td>Expands an edge towards the outside of the polygon it originated from by the amount specified in the number1 parameter. This secondary keyword can be used along with INside BY.</td>
</tr>
<tr>
<td>OUTside BY FACTOR</td>
<td>Expands an edge towards the outside of the polygon it originated from by the product of the number1 parameter and the edge's length. This secondary keyword can be used along with INside BY.</td>
</tr>
<tr>
<td>BY</td>
<td>Performs expansion of both secondary keywords INside BY and OUTside BY. You cannot use this parameter with any other secondary keyword in this set.</td>
</tr>
<tr>
<td>BY FACTOR</td>
<td>Performs expansion of both secondary keywords INside BY FACTOR and OUTside BY FACTOR. You cannot use this parameter with any other secondary keyword in this set.</td>
</tr>
<tr>
<td>number1</td>
<td>A positive real number, must follow the specified secondary keywords listed for expansion_set.</td>
</tr>
<tr>
<td>[EXTEND BY [FACTOR] number 2]</td>
<td>Extends or retracts both ends of an edge prior to expanding into rectangles. A positive number2 parameter specifies the amount to extend both ends of an edge. A negative number2 parameter specifies the amount to retract both ends of an edge. An edge disappears if it is retracted more than or equal to two times its original length</td>
</tr>
</tbody>
</table>
[CORNER FILL] An optional secondary keyword that directs an Expand Edge operation to fill gaps between rectangles formed by the operation at corners of the input layer. You cannot specify CORNER FILL with a BY FACTOR secondary keyword or an EXTEND secondary keyword.
EXPAND TEXT

**EXPAND TEXT** text_name [text_layer] BY number [PRIMARY ONLY]

**Description**

Produces a derived polygon layer containing squares centered on the locations of text objects having the specified text_name. The sides of the squares have length equal to number.

The parameter order must be observed to avoid ambiguity. The text_name is case insensitive. The text_name parameter can contain one or more question mark (?) wildcard characters, where the (?) matches zero or more characters.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>text_name</strong></td>
<td>The name of a text object. Can be a string variable.</td>
</tr>
<tr>
<td><strong>text_layer</strong></td>
<td>An optional drawn layer that contains the text_name. The text_layer option is used to restrict the text search to the specified layer.</td>
</tr>
<tr>
<td><strong>number</strong></td>
<td>A positive floating-point number in user units that specifies the size of the squares.</td>
</tr>
<tr>
<td><strong>PRIMARY ONLY</strong></td>
<td>An optional keyword that specifies only text at the top level of the hierarchy is used.</td>
</tr>
</tbody>
</table>
EXT

EXTernal layer1 constraint [metric] [polygon_filter] [polygon_containment] [connectivity_filter] [orientation_filter] [projection_filter] [angled_filter] [corner_filter] [intersection_filter] [reversal] [output]

EXTernal layer1 layer2 constraint [metric] [polygon_containment] [connectivity_filter] [orientation_filter] [projection_filter] [angled_filter] [corner_filter] [intersection_filter] [reversal] [output]

Description

Measures the distance between the outside of layer1 boundaries, or the distance between the outside of layer1 and the outside of layer2 boundaries. Edge pairs that meet the constraint are output. By default, intersecting or overlapping edge pairs are not compared.

Parameters

The following parameters are common to all ENC, EXT and INT rule checks.

layer1
A drawn layer, derived polygon layer, or derived edge layer.

layer2
A drawn layer, derived polygon layer, or derived edge layer.

Constraint
Constraint is any range defined by “Constraints” (page 491), except >a, >=a, or !=a, where the constraints values must be a non-negative real number in user units.

metric
EUCLIDEAN (Default) — Use the euclidean metric.

OPPOSITE — Use the opposite metric.

SQUARE — Use the square metric.

polygon_containment
MEASURE ALL — Specifies that all edges are compared, including those normally not compared due to intersection, polygon containment, or visibility blocking.

MEASURE COINCident — Specifies that edge pairs that are outside coincident that would normally not be compared, should be compared.

connectivity_filter
CONNECTED — Specifies that only edges from the same net are compared.

NOT CONNECTED — Specifies that only edges from different nets are compared.
orientation_filter

acute_filter

ACUTE ALSO — Specifies to measure edges with an appropriate angle between 0 and 90 degrees, exclusive. This is the default behavior if you do not specify a choice from the acute_filter option subset.

ACUTE ONLY — Specifies to measure only edges with an appropriate angle between 0 and 90 degrees, exclusive. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same External operation.

NOT ACUTE — Specifies to not measure edges with an appropriate angle between 0 and 90 degrees.

parallel_filter

PARAllel ALSO — Specifies to measure parallel edges. This is the default behavior if you do not specify a choice from the parallel_filter option subset.

PARAllel ONLY — Specifies to measure only parallel edges. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same External operation.

NOT PARAllel — Specifies to not measure parallel edges.

perpendicular_filter

NOT PERPendicular — Specifies to not measure perpendicular edges. This is the default behavior if you do not specify a choice from the perpendicular_filter option subset.

PERPendicular ONLY — Specifies to measure only perpendicular edges. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same External operation.

PERPendicular ALSO — Specifies to measure perpendicular edges.

obtuse_filter

NOT OBTUSE — Specifies to not measure edges with an appropriate angle between 90 and 180 degrees, exclusive. This is the default behavior if you do not specify a choice from the obtuse_filter option subset.

OBTUSE ONLY — Specifies to measure only edges with an appropriate angle between 90 and 180 degrees, exclusive. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same External operation.

OBTUSE ALSO — Specifies to measure edges with an appropriate angle between 90 and 180 degrees, exclusive.
**projection_filter**

**PROJecting [constraint]** — Specifies to compare the distance between two edges only when one edge projects onto the other edge and the length of projection conforms to the constraint. This is the default behavior, with constraint set to “>=0”, if you do not specify a choice from this set in the operation.

**NOT PROJecting** — Specifies to compare the distance between two edges only when neither edge projects onto the other edge.

**angled_filter**

**ANGLED [constraint]** — Specifies to measure the two edges only when the number of non-orthogonal (angled) edges in the pair meets the given constraint. The constraint is optional and when omitted, defaults to “> 0”.

**corner_filter**

You cannot specify this filter with an orientation, projection, or angled filter.

**CORNER TO CORNER** — Specifies to measure and output errors only for edges in a corner-to-corner configuration.

Additional options **CORNER, CORNER TO EDGE, and NOT CORNER** are not supported at this time.
**intersection_filter**

Specifies to measure intersecting edges, and specifies exactly the characteristics of intersecting edges that are to be measured. Intersecting or abutting (coincident) edges are not measured by default. Coincident or abutting edges are considered to be intersecting. The value of intersection_filter is:

```
[ABUT [abut_constraint] ] [OVERLAP] [SINGULAR]
[INTERSECTING ONLY]
```

You can specify any combination of ABUT, OVERLAP, and SINGULAR in one operation.

**ABUT [abut_constraint]** — Specifies that intersecting edges should also be output if the angle between them conforms to the optional constraint (interpreted in degrees). Output from the ABUT condition is in addition to any other output the External operation generates. The abut_constraint modifier must contain non-negative real numbers less than 180. Single-operator constraints such as $< 90$ and $> 135$ are interpreted as $>= 0 < 90$ and $> 135 < 180$. With no constraint specified, the default value is $>= 0 < 180$.

If the abut_constraint modifier includes zero in its range then any edges A of layer1 which are coincident outside with edges B of layer2 are also output (because the angle between the exterior side of A and the exterior side of B is zero).

**OVERLAP** — Specifies that output should also occur where a polygon from one input layer crosses a polygon from the other input layer. Edges forming the point of overlap are measured and output as if an unconstrained ABUT parameter was specified. This overrides any specified ABUT parameter at the point of overlap only.

This keyword cannot be used when either of the input layers is a derived edge layer because polygons are required to determine if an overlap condition is present. Output from the overlap condition is in addition to any other output generated by the Enclosure operation.

**SINGULAR** — Specifies that intersecting edges at points of polygon singularity should also be output. Singularities are point-to-edge or point-to-point polygon intersections or self-intersections.

**INTERSECTING ONLY** — Specifies that only intersecting edge errors are to be output. Must be preceded with at least one of the other options in this group. The ABUT, OVERLAP, and SINGULAR options normally add intersecting edge violations to the non-intersecting edge violations that are normally output. The INTERSECTING ONLY option, when added to one of the other options, causes only intersecting edge violations to be output. This filter ignores orientation, angled, projection, and corner filters.

**Output**

**REGION [EXTENTS]** — constructs polygon projections between the error edges and then outputs them as polygon data. When you specify REGION the polygon region is used, but when you specify REGION EXTENTS the extents of the polygon region is used.
The following parameters apply to EXT only.

**reversal**

**INSIDE ALSO** — outputs edges and edge segments from layer1 or layer2 that are inside or inside-coincident of the other layer.

The connectivity keywords CONNECTED and NOT CONNECTED do affect the output from the INSIDE ALSO option. If you specify NOT CONNECTED, then the INSIDE ALSO parameter does not output an edge if the polygon containing it (or the edge coincident-inside with it) are on the same electrical node. If you specify CONNECTED, then the INSIDE ALSO parameter does not output an edge if the polygon containing it (or the edge coincident inside with it) are on different electrical nodes.

The following parameters apply to **single layer** EXT only.

**polygon_filter**

The secondary keywords in this set instruct the one-layer External operation to measure the separation between the outsides of edges based upon polygon membership.

**NOTCH** — measures the separation between the outsides of two edges from only the same polygon.

**SPACE** — measures the separation between the outsides of two edges from only different polygons.

If you do not specify either NOTCH or SPACE, then both conditions are measured.

**Examples**

```
ACT.S.1 { @ Minimum ACTIVE spacing < 0.5
         EXT ACTIVE< 0.5 SINGULAR ABUT <90
    }
```
EXTENT

[polygon-layer =] EXTENT [layer1]

Description

Produces a layer consisting of one rectangle equal to the minimum bounding box of all input layers. If the layer1 option is present, then the resulting polygon layer consists of one rectangle equal to the minimum bounding box of that layer.

Parameters

layer1  A drawn or derived polygon layer

Examples

CHIP = EXTENT
EXTENTS

[polygon-layer =] EXTENTS layer1 [CENTERS [number]]

Description

Produces a layer consisting of the (merged) minimum bounding boxes of each polygon on layer1.

Parameters

layer1  A drawn or derived polygon layer
CENTERS  Produce squares at the center of each bounding box instead of the bounding boxes themselves.
number  Specifies the size of the generated squares if CENTERS is specified. The square will have the dimension of number user units by number user units. The default value is 1.

Examples

// Center-to-center pad distance must be 200 microns:

pad_center = EXTENTS pad CENTERS 2
pad_spacing {
  EXT pad_center < 198 // Use 198 since centers are 2x2
}
FLAG ACUTE

FLAG ACUTE {NO | YES}

Description

Reports an error for any two consecutive edges of a drawn polygon or wire that form an acute angle. Errors will appear in the Error Navigator with a rule name “Polygons and wires with acute angles (< 90°)”. This command can appear only once in the command file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Default. Do not report acute angles.</td>
</tr>
<tr>
<td>YES</td>
<td>Report acute angles.</td>
</tr>
</tbody>
</table>

Examples

FLAG ACUTE YES
FLAG NONSIMPLE

FLAG NONSIMPLE YES

Description

Reports an error if two edges on the same drawn polygon intersect or if filled region of the polygon is ambiguous. Wires are checked for intersections based on the outer boundary of the wire.

Self intersecting polygons and wires are ignored by all layer generation and rule checking commands. Self-intersecting polygons and wires are always reported, this command is not required. A “NO” argument in this command will be ignored, and self intersections will be checked anyway. This command can appear only once in the command file.
FLAG OFFGRID

FLAG OFFGRID {NO | YES}

Description

Report an error for every offgrid vertex of drawn polygons and wires. Also report errors for offgrid instance placements, rotated instances, and instance arrays and scaling that could result in offgrid geometry. Checking and reporting is done in the context of the cell. The grid is defined by the RESOLUTION command. Errors will appear in the Error Navigator with a rule name “Offgrid (grid size) objects”. This command can appear only once in the command file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Default. Do not report offgrid vertices and instances.</td>
</tr>
<tr>
<td>YES</td>
<td>Report offgrid vertices and instances.</td>
</tr>
</tbody>
</table>

Examples

FLAG OFFGRID YES
FLAG POLYGONVERTEXLIMIT

FLAG POLYGONVERTEXLIMIT {NO | maxvertices}

**Description**

Reports an error for any drawn polygon with more than a specified number of vertices. This command can appear only once in the command file.

**Parameters**

- **NO**
  - Default.
- **maxvertices**
  - Drawn polygons with more than maxvertices are reported as an error.
  - A positive integer.

**Examples**

Polygons in GDSII files have the last vertex repeat the first vertex, so to flag polygons that will have more than 200 vertices when your file is saved to GDS, set maxvertices equal to 199.

FLAG POLYGONVERTEXLIMIT 199

**Note:**

A Calibre flag for FLAG POLYGONVERTEXLIMIT does not exist, the command is an extension that Tanner supports because it corresponds to one of our geometry flags in L-Edit Standard DRC.
FLAG SKEW

FLAG SKEW {NO | YES}

Description

Reports an error for any non-90 or non-45 degree edge of a drawn polygon or centerline segment of a drawn wire. Reporting is based on the angle of the edge in its cell coordinate space, not in coordinate space of instances of that cell. Errors will appear in the Error Navigator with a rule name “All-angle edges”. This command can appear only once in the command file.

Parameters

- NO: Default. Do not report skew edges
- YES: Report skew edges

Examples

FLAG SKEW YES
FLAG WIREVERTEXLIMIT

FLAG WIREVERTEXLIMIT {NO | maxvertices}

Description

Reports an error for any wire with more than a specified number of vertices. This command can appear only once in the command file.

Parameters

- **NO**
  - Do not report wire vertex count errors.

- **maxvertices**
  - Wires with more than maxvertices are reported as an error. A positive integer.

Examples

FLAG WIREVERTEXLIMIT 200

Note: A Calibre flag for FLAG WIREVERTEXLIMIT does not exist, the command is an extension that Tanner supports because it corresponds to one of our geometry flags in L-Edit Standard DRC.
FLAG ZEROWIDTHWIRES

FLAG ZEROWIDTHWIRES {NO | YES}

Description

Reports an error for any zero width wire. This command can appear only once in the command file.

Parameters

NO
Default. Do not report zero width wires.

YES
Report zero width wires.

Examples

FLAG ZEROWIDTHWIRES YES

Note:
A Calibre flag for FLAG ZEROWIDTHWIRES does not exist, the command is an extension that Tanner supports because it corresponds to one of our geometry flags in L-Edit Standard DRC.
FLATTEN

[result-layer =] FLATTEN layer1

Description

Flattens the specified layer to the top level.

Parameters

layer

A drawn or derived polygon layer or a derived edge layer.
FLATTEN CELL

FLATTEN CELL name ...

Description

Flattens instances of specified cells into their parent. Instances of the specified cells are replaced by their flat contents. This operation automatically finds or matches cells that come from T-Cell and X-Ref cells.

Parameters

name

The name of a cell. If a cell name has spaces, then the name is in quotes. Name can be specified any number of times in one statement.

The cell name parameters can contain one or more asterisk (*) wildcard characters, where the (*) matches zero or more characters. When using the * character, enclose the cell name in quotes. Otherwise, a compilation error occurs because the asterisk is a reserved symbol.

Examples

FLATTEN CELL cell0 cell1
**FLATTEN INSIDE CELL**

**FLATTEN INSIDE CELL** name ...

*Description*

Flattens the contents of specified cells. Instances of the specified cells remain in place, with their contents flattened. Cells may also be identified for flattening using Tools > Add Inst > Mark cells for flattening during DRC. This operation automatically finds or matches cells that come from T-Cell and X-Ref cells.

*Parameters*

- **name**
  
  The name of a cell. If a cell name has spaces, then the name is in quotes. Name can be specified any number of times in one statement. Wildcards are not permitted at this time.

*Examples*

Consider cell A which contains instances of cell B, and cell A is instantiated in cell Top. FLATTEN INSIDE CELL A will cause the contents of A to be flattened, but there will still be instances of A in cell Top.
GROUP

GROUP name rule_check ...

Description

Names a set of RuleCheck statements. The group name can then be used in DRC SELECT CHECK and
DRC UNSELECT CHECK commands.

The (?) character is a wildcard that matches zero or more characters. Names in the body of a Group
statement can contain one or more question mark (?) characters. Wildcard matching is only applied to
RuleCheck statement names in the group definition, and not to other group names.

Parameters

name
rule_check

A text string that specifies the name of the RuleCheck group.
A RuleCheck name or group name.

Examples

GROUP POLY_SPACING_RULES PO.S1 PO.S2 PO.S3
DRC SELECT CHECK POLY_SPACING_RULES

The above GROUP statement can be written more simply using the (?) wildcard character:

GROUP POLY_SPACING_RULES PO.S?
GROW

[polygon-layer =] GROW layer1 [RIGHT BY value] [TOP BY value] [LEFT BY value] [BOTTOM BY value]

Description

Performs outward translation of the input layer edges in the direction of the x-axis, y-axis, or both.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>An drawn or derived polygon layer or derived edge layer.</td>
</tr>
<tr>
<td>RIGHT BY value</td>
<td>Translate the right edge(s) of each polygon on layer1, toward the outside of each polygon layer1, by the specified value. Value is a non-negative real number in user units.</td>
</tr>
<tr>
<td>TOP BY value</td>
<td>Translate the top edge(s) of each polygon on layer1, toward the outside of each polygon layer1, by the specified value. Value is a non-negative real number in user units.</td>
</tr>
<tr>
<td>LEFT BY value</td>
<td>Translate the left edge(s) of each polygon on layer1, toward the outside of each polygon layer1, by the specified value. Value is a non-negative real number in user units.</td>
</tr>
<tr>
<td>BOTTOM BY value</td>
<td>Translate the bottom edge(s) of each polygon on layer1, toward the outside of each polygon layer1, by the specified value. Value is a non-negative real number in user units.</td>
</tr>
</tbody>
</table>
HOLES

[polygon-layer =] HOLES layer1 [constraint] [INNER] [EMPTY]

Description

Produces a layer consisting of polygons that exactly fit inside the holes of layer1 polygons. Optionally, only produce polygons whose area satisfies a constraint.

Parameters

layer1 A drawn or derived polygon layer.
constraint Constraint is any range defined by “Constraints” (page 491).
inner if a hole contains another hole, only produce the inner hole.
empty Prevents output of holes that contain polygons on the input layer. Specifically prevent output of holes that are NOT OUTSIDE the input layer.
INCLUDE

INCLUDE filename...

Description

Includes the specified command file into the current command file.

Parameters

| filename   | A valid path and filename. Relative paths are resolved with respect to the location of the topmost command file. |
INSIDE

\[ \text{polygon-layer} = \text{layer1} \ [\text{NOT}] \ \text{INSIDE} \ \text{layer2} \]
\[ \text{polygon-layer} = \ [\text{NOT}] \ \text{INSIDE} \ \text{layer1} \ \text{layer2} \]

**Description**

Produces all layer1 polygons that are completely inside layer2 polygons. Touching from the inside is considered to be inside. The NOT option produces those layer1 polygons not produced by the corresponding INSIDE operation.

**Parameters**

- **layer1**: A drawn or derived polygon layer
- **layer2**: A drawn or derived polygon layer
INSIDE CELL

[NOT] INSIDE CELL layer1 cellname [... cellname] [PRIMARY ONLY]

Description

Produces all polygons from the input layer that are contained inside the specified cells, and the sub-hierarchies of those cells. PRIMARY ONLY causes only top-level geometry from specified cells to be produced. Parameters must appear in the specified command order to avoid ambiguity.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn layer.</td>
</tr>
<tr>
<td>cellname</td>
<td>The name of a cell. Any number of cell names can be specified, separated by a space, and cell names are case-sensitive. The name can be a string variable (see Variable). The cellname can contain one or more asterisk (*) wildcard characters, where the * character matches zero or more characters. When using *, enclose the cell name in quotes because the asterisk is a reserved symbol.</td>
</tr>
<tr>
<td>PRIMARY ONLY</td>
<td>When PRIMARY ONLY is specified, only geometry in the top level of the specified cells is output. Geometry from the sub-hierarchy of a specified cell may be output if it is the top level of another specified cell.</td>
</tr>
</tbody>
</table>
INSIDE EDGE

[edge-layer =] [NOT] INside EDGE layer1 layer2
[edge-layer =] layer1 [NOT] INside EDGE layer2

**Description**

Produces all layer1 edge segments that are completely contained inside layer2 polygons. The NOT option produces all layer1 edges not produced by the corresponding INSIDE EDGE operation.

**Parameters**

- **layer1**: A drawn or derived polygon layer or a derived edge layer.
- **layer2**: A drawn or derived polygon layer.
INT

INTernal layer1 constraint [metric] [orientation_filter] [projection_filter]
[angled_filter] [corner_filter] [intersection_filter] [output]

INTernal layer1 layer2 constraint [metric] [polygon_containment]
[connectivity_filter] [orientation_filter] [projection_filter]
[angled_filter] [corner_filter] [intersection_filter] [output]

Description

Measures the distance between the inside of layer1 boundaries, or the inside of layer1 and the inside of layer2 boundaries. Edge pairs that meet the constraint are output. By default, intersecting or overlapping edge pairs are not compared.

Parameters

The following parameters are common to all ENC, EXT and INT RuleChecks.

layer1
A drawn layer, derived polygon layer, or derived edge layer.

layer2
A drawn layer, derived polygon layer, or derived edge layer.

Constraint
Constraint is any range defined by “Constraints” (page 491), except > a, >= a, or != a, where the constraints values must be a non-negative real number in user units.

metric
EUCLIDEAN (Default) — Use the euclidean metric.

OPPOSITE — Use the opposite metric.

SQUARE — Use the square metric.

polygon_containment
MEASURE ALL — Specifies that all edges are compared, including those normally not compared due to intersection, polygon containment, or visibility blocking.

MEASURE COINCident — Specifies that edge pairs that are outside coincident that would normally not be compared, should be compared.

connectivity_filter
CONNECTED — Specifies that only edges from the same net are compared.

NOT CONNECTED — Specifies that only edges from different nets are compared.
**orientation_filter**

**acute_filter**

**ACUTE ALSO** — Specifies to measure edges with an appropriate angle between 0 and 90 degrees, exclusive. This is the default behavior if you do not specify a choice from the acute_filter option subset.

**ACUTE ONLY** — Specifies to measure only edges with an appropriate angle between 0 and 90 degrees, exclusive. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same Internal operation.

**NOT ACUTE** — Specifies to not measure edges with an appropriate angle between 0 and 90 degrees.

**parallel_filter**

**PARAllel ALSO** — Specifies to measure parallel edges. This is the default behavior if you do not specify a choice from the parallel_filter option subset.

**PARAllel ONLY** — Specifies to measure only parallel edges. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same Internal operation.

**NOT PARAllel** — Specifies to not measure parallel edges.

**perpendicular_filter**

**NOT PERPendicular** — Specifies to not measure perpendicular edges. This is the default behavior if you do not specify a choice from the perpendicular_filter option subset.

**PERPendicular ONLY** — Specifies to measure only perpendicular edges. You cannot use this parameter and a parameter from the other subsets of orientation_filter in the same Internal operation.

**PERPendicular ALSO** — Specifies to measure perpendicular edges.
orientation_filter
(continued)

obtuse_filter

**NOT OBTUSE** — Specifies to not measure edges with an appropriate angle between 90 and 180 degrees, exclusive. This is the default behavior if you do not specify a choice from the obtuse_filter option subset.

**OBTUSE ONLY** — Specifies to measure only edges with an appropriate angle between 90 and 180 degrees, exclusive. You cannot use this parameter and an parameter from the other subsets of orientation_filter in the same Internal operation.

**OBTUSE ALSO** — Specifies to measure edges with an appropriate angle between 90 and 180 degrees, exclusive.

projection_filter

**PROJECTing [constraint]** — Specifies to compare the distance between two edges only when one edge projects onto the other edge and the length of projection conforms to the constraint. This is the default behavior, with constraint set to “>=0”, if you do not specify a choice from this set in the operation.

**NOT PROJECTing** — Specifies to compare the distance between two edges only when neither edge projects onto the other edge.

angled_filter

**ANGLED [constraint]** — Specifies to measure the two edges only when the number of non-orthogonal (angled) edges in the pair meets the given constraint. The constraint is optional and when omitted, defaults to “> 0”.

You cannot specify this filter with an orientation, projection, or angled filter.

**CORNER TO CORNER** — Specifies to measure and output errors only for edges in a corner-to-corner configuration.

Additional options **CORNER, CORNER TO EDGE, and NOT CORNER** are not supported at this time.
intersection_filter

Specifies to measure intersecting edges, and specifies exactly the characteristics of intersecting edges that are to be measured. Intersecting or abutting (coincident) edges are not measured by default. Coincident or abutting edges are considered to be intersecting. The value of intersection_filter is:

\[\text{ABUT} \ [\text{abut_constraint}] \ [\text{OVERLAP}] \ [\text{SINGULAR}] \ [\text{INTERSECTING ONLY}]\]

You can specify any combination of ABUT, OVERLAP, and SINGULAR in one operation.

**ABUT [abut_constraint]** — Specifies that intersecting edges should also be output if the angle between them conforms to the optional constraint (interpreted in degrees). Output from the ABUT condition is in addition to any other output the Internal operation generates. The abut_constraint modifier must contain non-negative real numbers less than 180. Single-operator constraints such as \(< 90\) and \(> 135\) are interpreted as \(\geq 0 < 90\) and \(> 135 < 180\). With no constraint specified, the default value is \(\geq 0 < 180\).

If the abut_constraint modifier includes zero in its range then any edges A of layer1 which are coincident outside with edges B of layer2 are also output (because the angle between the exterior side of A and the interior side of B is zero).

**OVERLAP** — Specifies that output should also occur where a polygon from one input layer crosses a polygon from the other input layer. Edges forming the point of overlap are measured and output as if an unconstrained ABUT parameter was specified. This overrides any specified ABUT parameter at the point of overlap only.

This keyword cannot be used when either of the input layers is a derived edge layer because polygons are required to determine if an overlap condition is present. Output from the overlap condition is in addition to any other output generated by the Enclosure operation.

**SINGULAR** — Specifies that intersecting edges at points of polygon singularity should also be output. Singularities are point-to-edge or point-to-point polygon intersections or self-intersections.

**INTERSECTING ONLY** — Specifies that only intersecting edge errors are to be output. Must be preceded with at least one of the other options in this group. The ABUT, OVERLAP, and SINGULAR options normally add intersecting edge violations to the non intersecting edge violations that are normally output. The INTERSECTING ONLY option, when added to one of the other options, causes only intersecting edge violations to be output. This filter ignores orientation, angled, projection, and corner filters.

**Output**

**REGION [EXTENTS]** — constructs polygon projections between the error edges and then outputs them as polygon data. When you specify REGION the polygon region is used, but when you specify REGION EXTENTS the extents of the polygon region is used.
Examples

M1.W.1 { @ Minimum M1 width < 0.6
   INT M1 < 0.6 SINGULAR REGION ABUT < 90
INTERACT

\[ \text{[polygon-layer =]} \text{ layer1 [NOT] INTERACT layer2 [constraint [BY NET]]} \]
\[ \text{[polygon-layer =]} \text{ [NOT] INTERACT layer1 layer2 [constraint [BY NET]]} \]

**Description**

Produces all layer1 polygons that have some or all area inside layer2 polygons or share an edge with layer2 polygons. The NOT option produces those layer1 polygons not produced by the corresponding INTERACT operation. The input layers will be merged if the constraint option is used, and will run slower than the INTERACT operation with no constraint.

(This option is not yet supported.) If BY NET is specified, the constraint applies to the number of layer2 polygons on distinct nets that CUT layer1.

**Parameters**

- **layer1**: A drawn or derived polygon layer
- **layer2**: A drawn or derived polygon layer
- **constraint**: Specifies the number of layer2 polygons that a layer1 polygon must interact with, in order to be selected by the INTERACT operation. Constraint is any range defined by “Constraints” (page 491) where the constraints values must be integer.
- **BY NET**: When the BY NET option is used, then a layer1 polygon is selected when the specified number of layer2 polygons on distinct nets interact the layer1 polygon. Layer2 must have connectivity.
LABEL ORDER

LABEL ORDER layer [... layer]

Description

LABEL ORDER defines the layer sequence in which polygons on a net are examined for intersection and attachment with a net naming label. Only layers listed in the TEXT LAYER command are used in net naming. The process of assigning label names to nets gives first priority to explicit ATTACH commands, then to implicit attachment (labels on same layer as overlapping polygon), and then to the LABEL ORDER for the remaining text labels. When multiple polygons on different nets overlap a label, then the net with the polygon whose layer appears first in the Label Order list is labeled with the value of the net name.

Parameters

layer A drawn layer or a derived polygon layer. This layer must appear as an input layer to a Connect or Sconnect operation in the same verification set. You can specify layer any number of times in one statement.


**LAYER**

```
LAYER name {GDS# ... | drawn_layer ...}
```

**Description**

 Specifies the name of a drawn layer in terms of its GDSII number or other previously defined names.

 A LAYER statement is required in the command file in order to use a drawn layer in a layer operation or DRC command. LAYER statements can be used to map a layer name in the L-Edit editing environment to a different name in the DRC command file by assigning the same GDSII number to the layer in each location.

 To make a layer in the tdb file to be equal to different layer name in the DRC command file then assign the same GDS number to that layer in Setup Layers > General in the tdb file, and to the new name in a LAYER statement in the command file.

**Parameters**

- **name**
  - A drawn layer name.

- **GDS#**
  - A list of GDS numbers.

- **drawn_layer**
  - A list of previously defined layer names.

**Examples**

```
LAYER Active 3 11 12
LAYER Poly1 46
LAYER Poly2 47
LAYER AllPoly Poly1 Poly2
```
LAYER MAP

\[
\text{LAYER MAP source\_layer \{DATATYPE | TEXTTYPE\} source\_type target\_layer}
\]

Description

Specifies a mapping of specified GDS number and DATATYPE or TEXTTYPE to different layers.

Parameters

- **source\_layer**: A positive integer or a constraint of integers that represents the GDS layer number of the source layer. An integer specifies a single source layer, and a constraint specifies a range of source layers.

- **\{DATATYPE | TEXTTYPE\}**: Use DATATYPE to map geometry and TEXTTYPE to map text (ports).

- **source\_type**: A positive integer or a constraint of integers that represents the GDS datatype of the source layer. An integer specifies a single datatype, and a constraint specifies a range of datatypes.

- **target\_layer**: A positive integer that specifies the GDS number of the target layer.

When using the LAYER MAP command in a command file, there should be corresponding layers and datatypes setup in L-Edit. Use *Setup > Layers* to setup GDS layer numbers and datatypes for the layout. The datatypes in the Setup Layers dialog are used to initialize the datatype of new objects created on a layer. Assigning a datatype to a layer will not automatically assign that datatype to pre-existing objects on that layer. You can propagate the layer datatype to all existing objects on that layer using *Draw > Assign GDSII Datatypes*.

Examples

// Layers M1 and M1\_TIGHT both have GDS number 49 in L-Edit, // but they have different datatypes. If we use *Setup > Layers* to setup GDS layer numbers and datatypes for the layer. The datatypes in the Setup Layers dialog are used to initialize the datatype of new objects created on a layer. Assigning a datatype to a layer will not automatically assign that datatype to pre-existing objects on that layer. You can propagate the layer datatype to all existing objects on that layer using *Draw > Assign GDSII Datatypes*.

```plaintext
LAYER MAP 49 DATATYPE 0 101 // M1 layer
LAYER M1 101
LAYER MAP 49 DATATYPE 1 102 // M1\_TIGHT layer
LAYER M1\_TIGHT 102

M1.Spacing {
  EXT M1 < 3.0
}

M1\_TIGHT.Spacing {
  EXT M1\_TIGHT < 2.0
}
```


**LAYER RESOLUTION**

```
LAYER RESOLUTION layer1 {s | sx sy}
```

**Description**

Overrides the default RESOLUTION specification statement parameters for a specified drawn layer during off-grid vertex checking with Flag Offgrid, Drawn Offgrid, or Snap Offgrid. During off-grid checking, all geometries on layer1 are checked using the resolution specified in the Layer Resolution statement rather than the default specified in the Resolution specification statement.

The statement may be specified any number of times but only once for any particular drawn layer.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>layer1</strong></td>
<td>The name of a drawn layer (do not specify a layer set name).</td>
</tr>
<tr>
<td><strong>s</strong></td>
<td>A positive integer in database units that specifies the x-direction and y-direction grid size for off-grid checking for the specified layer.</td>
</tr>
<tr>
<td><strong>sx</strong></td>
<td>A positive integer in database units that specifies the x-direction grid size for off-grid checking for the specified layer.</td>
</tr>
<tr>
<td><strong>sy</strong></td>
<td>A positive integer in database units that specifies the y-direction grid size for off-grid checking for the specified layer.</td>
</tr>
</tbody>
</table>
LENGTH

[edge-layer =] [NOT] LENGTH layer1 constraint

Description

Produces all layer1 edges whose length conforms to the constraint. The NOT option produces all layer1 edges not produced by the corresponding LENGTH operation.

Parameters

layer1
A drawn or derived polygon layer or derived edge layer.

constraint
Constraint is any range defined by “Constraints” (page 491) where the constraints values must be a non-negative real number in user units.
MERGE

[result-layer =] MERGE layer1 [BY 0]

Description

Promotes polygons on the specified layer up the hierarchy and merges them such that polygons on the resultant layer are completely contained in a single cell, and are not spread over the hierarchy.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer</td>
<td>A drawn or derived polygon layer or a derived edge layer.</td>
</tr>
<tr>
<td>BY 0</td>
<td>MERGE layer1 BY 0 does not merge polygons in different cells that Abut, only merges polygons that overlap</td>
</tr>
</tbody>
</table>
NET

[NOT] NET layer1 net_name [...net_name]

Description

Produces all layer1 polygons that belong to the net having the specified netname. The connectivity on layer1 must be established through a connectivity operation.

Parameters

<table>
<thead>
<tr>
<th>layer1</th>
<th>A drawn layer or layer set, or a derived polygon layer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>net_name</td>
<td>A name of a net, which can contain one or more question characters. The ? is a wildcard character that matches zero or more characters. You can specify net_name any number of times in one statement. The net_name can also be a string variable (see Variable).</td>
</tr>
</tbody>
</table>
NET AREA

**NET AREA layer constraint**

**Description**

Selects all polygons on a specified layer that have a total area conforming to a constraint or a single net.

**Parameters**

- **layer1**: A drawn layer, layer set, or a derived polygon layer.
- **constraint**: A required constraint, which must be a non-negative floating-point number and is interpreted in user units squared. (See “Constraints” on page 491.)

**See Also**

“NET” (page 545), “CONNECT” (page 541), “SCONNECT” (page 546) and “STAMP” (page 548).
Chapter 20: HiPer Verify: Calibre Command Files

NET AREA RATIO

**Standard Form**

```
NET AREA RATIO
{layer1 [SCALE BY value] {[COUNT ONLY] | [PERIMETER ONLY]}}
[...{layerN [SCALE BY value] {[COUNT ONLY] | [PERIMETER ONLY]}}]
[OVER]
{d_layer1 [SCALE BY value] {[COUNT ONLY] | [PERIMETER ONLY]}}
[...{d_layerN [SCALE BY value] {[COUNT ONLY] | [PERIMETER ONLY]}}]
['[expression]'] constraint [INSIDE OF LAYER layer][ACCUMULATE [alayer]]
{{RDB file_name [BY LAYER]}
[{{rdb_layer [MAXIMUM max_polygon]}} {{rdb_layer [MAXIMUM max_polygon
...]]}}]
```

**Single-layer Form**

```
NET AREA RATIO layer [expression] constraint [INSIDE OF LAYER layer]
{{RDB file_name [BY LAYER]}
[{{rdb_layer [MAXIMUM max_polygon]}} {{rdb_layer [MAXIMUM max_polygon
...]]}}]
```

**ACCUMULATE-only Form**

```
NET AREA RATIO ACCUMULATE layer1 layer2 constraint [Aexpression]
```

**Description**

Selects polygons based on the ratios of polygon areas, perimeters, and counts on the same nets, primarily for antenna checks. The Standard form is the most general of the three. The Single-layer form is used for finding net information about a single layer. The ACCUMULATE-only form performs ratio accumulation calculations for two input layers. Note that the expression **INSIDE OF LAYER** is not supported.

**Standard Form**

The Standard form operation selects all polygons from layer1 that lie on an electrical node, such that the ratio of the total area of layerN polygons on that node to the total area of d_layerN polygons satisfies the given constraint. The term “area” is used loosely here, as the calculated ratio can pertain to other quantities like polygon counts or perimeters. The unit dimensions of the ratio that is calculated are determined by the [expression], if provided; otherwise, area is used as the unit of measurement.

Note that all layer1 polygons on the node are selected if the ratio meets the constraint. The behavior is different if you use the ACCUMULATE, rdb_layer, or BY LAYER parameters. For ACCUMULATE, d_layer1 polygons are selected. For rdb_layer (and BY LAYER), you can specify input layers other than layer1 or d_layer1, which can be selected for output.

**Single-layer Form**

An [expression] is required in the single-layer form. The input layer behaves like layer1 in the standard form (ACCUMULATE not specified) and like d_layer1 in the standard form (ACCUMULATE is specified). There is no == 0 constraint special case in the single-layer form. Otherwise, the semantics are identical to the standard form. For example:
// Derive a NARAC layer where each gate polygon has its
// individual area attached:
CONNECT gate
area_gate = NET AREA RATIO gate >= 0 [ AREA( gate ) ] ACCUMULATE

**ACCUMULATE-only Form**

Parameters layer1 and layer2 must each be the output layer of a previous Net Area Ratio ACCUMULATE operation. In addition, they must have the same layer of origin, just as d_layer1 and alayer in the Standard form. The operation creates a Net Area Ratio ACCUMULATE layer consisting of a subset of the polygons in layer1 with potentially new values attached.

The [expression] is similar to the [expression] in the Standard form, except that the only function which may be computed for the input layers is VALUE, which returns the value attached to the polygon(s) for which the expression is computed.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer n</td>
<td>A drawn layer, layer set, or a derived polygon layer, which must have connectivity information. In Standard form you can layer as often as you like. In Single-layer form you can only specify one layer. In ACCUMULATE-only form you can only specify two input layers, which must be derived from previous ACCUMULATE-only operations.</td>
</tr>
<tr>
<td>OVER</td>
<td>This keyword must be included when you use more than one d_layer in Standard form.</td>
</tr>
<tr>
<td>d_layer n</td>
<td>A required drawn layer, layer set, or a derived polygon layer, which must have extracted connectivity information. These are the layers that are part of the ratio denominator. You can specify d_layer, along with any optional keywords, any number of times in one Standard form statement, but you must include the OVER keyword when you do so.</td>
</tr>
<tr>
<td>[expression]</td>
<td>A numeric expression that allows customizable control over the operation computations. The expression defines calculations to be made using the input layers. These calculations may or may not be ratios. They cannot produce strictly negative results. The expression must be contained in square brackets.</td>
</tr>
</tbody>
</table>

The [expression] is mandatory in the Single-layer form. For the ACCUMULATE-only form, the [expression] is mandatory but is evaluated differently than an [expression]. The only function available for this type of expression is VALUE. (See “ACCUMULATE-only Form” on page 557.)

The expression may contain numbers, numeric variables, binary operators (^, *, /, +, -), unary operators (+, -, !, ~), and algebraic or transcendental functions as follows:
[expression], continued

- ! — returns “0” (false) if its argument is non-zero and “1” (true) if its argument is zero. Used in front of AREA(), (most efficient), PERIMETER(), or COUNT(), the ! operator can be extremely valuable in performance optimizations for antenna checking where regions connected to diodes are not to be counted in antenna area calculations, for instance.

- ~ — returns “0” (false) if the argument is positive and “1” (true) if the argument is non-positive. The ~ operator is particularly useful where the sign of the argument is of concern.

- AREA(input_layer)
- PERIMETER(input_layer)
- COUNT(input_layer)
- SQRT(x) — square root of x
- EXP(x) — exponential (base e) of x
- LOG(x) — natural logarithm of x
- SIN(x) — sine of x radians
- COS(x) — cosine of x radians
- TAN(x) — tangent of x radians

**Note:** The input_layer is an input layer in the same Net Area Ratio statement and is not an ACCUMULATE layer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint</td>
<td>A required string that must contain non-negative real numbers (&lt; 0 is not allowed). It is interpreted as the constraining value of the operation.</td>
</tr>
<tr>
<td>INSIDE OF LAYER layer</td>
<td>An optional keyword and layer name set that causes the Net Area Ratio calculation for the input layerN polygons to be taken only for the portions of the layerN polygons that are inside polygons on the INSIDE OF LAYER layer parameter. The layer parameter must be an original or derived polygon layer. This keyword set is only available in a standard form or single-layer form operation and may not be used with the ACCUMULATE or RDB keyword sets.</td>
</tr>
<tr>
<td>ACCUMULATE [alayer]</td>
<td>An optional keyword set that attaches Net Area Ratio values to the derived output layer. The optional alayer is a derived layer from a previous Net Area Ratio ACCUMULATE operation (which specifies to add such derived output layer values from alayer to a new output layer.) The output layer can either be a derived layer or an error layer.</td>
</tr>
</tbody>
</table>
The basic syntax for NET AREA RATIO is:

\[
\text{NET AREA RATIO} \ layer1 \ [...] \ layerN \ [\text{OVER}] \ d\_layer1 \ [...] \ d\_layerN
\]
[expression]  constraint  [ACCUMULATE [alayer]]

Given the command

NET AREA RATIO $L_1 \ L_2 \ ... \ L_n$ OVER $D_1 \ D_2 \ ... \ D_m$ constraint

$L_1 \ L_2 \ ... \ L_n$ are numerator layers and $D_1 \ D_2 \ ... \ D_m$ are denominator layers. For each net, the NET AREA RATIO command computes the total area of all polygons on layers $L_1 \ ... \ L_n$ divided by the total area of all polygons on layers $D_1 \ ... \ D_m$, and outputs $L_1$ polygons if the computed ratio meets the constraint. If the NET AREA RATIO command contains an expression, then the expression is evaluated for each net, and $L_1$ polygons are output if the expression meets the constraint.

When ACCUMULATE is present, for example

NET AREA RATIO $L_1 \ L_2 \ ... \ L_n$ OVER $D_1 \ D_2 \ ... \ D_m$ constraint ACCUMULATE AC1

then $D_1$ polygons are output.

**Poly Check**

A simple field-poly check can be written as

Gate = Poly \ AND \ Active
CONNECT Poly Gate
ANT.Poly { @ Error if Area (Poly) / Area (Gate) > 100
            NET AREA RATIO Poly Gate > 100
}

ANT.Poly will flag an error if the ratio of the area of Poly to the area of Gate on any net exceeds 100. The output of the check are the Poly regions that could cause gate failure. The sidewall areas of Poly are ignored in this check.

**Metal Layer Check**

VARIABLE thickness 0.5
Gate = Poly \ AND \ Active
CONNECT Poly Gate
CONNECT Metal1 Poly BY PolyCont

ANT.Metal1 { @ Error if (sidewall area of Metal1) / (area of gate) > 100
             NET AREA RATIO Metal1 Gate >100
             [PERIMETER(Metal1)*thickness/AREA(Gate)]
}

ANT.Metal1 will flag an error if the ratio of the sidewall area of Metal1 to the area of Gate on a net exceeds 100. The output of the check are the Metal1 regions that could cause gate failure. The top areas of Metal1 are ignored in this check.

**Net Area Accumulation Ratio**

Antenna checks at each level of interconnect may not be sufficient to accurately check for charge accumulation. Using INCREMENTAL CONNECT YES and the ACCUMULATE option of NET AREA RATIO, you can model and check the accumulation of charge over multiple interconnect layers.
The areas of the polygons are indicated in the following figure:

The command file below models the accumulation of charge on gates through the fabrication of Metal1, Metal2, and Metal3 interconnect layers. Area ratios of metal to gate are computed and accumulated on gates through the fabrication of the interconnect layers, and gates whose accumulated area ratios exceed 150 are flagged as violations.

TITLE "Calibre NET AREA RATIO with ACCUMULATE"

DRC INCREMENTAL CONNECT YES

Gate = Poly AND Active

CONNECT Gate Poly
CONNECT Metall Poly BY PolyCont
	/*************************
	/***** Connectivity zone 1 ***
	/*************************

// AC1 contains all Gate polygons, with the Metall/Gate ratio
// attached to gate polygons.
AC1 = NET AREA RATIO Metall Gate >= 0 ACCUMULATE

// Write out Metall/Gate area ratios for debug
NET AREA RATIO PRINT AC1 NAR_AC1.txt

At this point AC1 contains all Gate polygons, with the Metall/Gate ratio attached to gate polygons.
The ratios of Metal1 area to gate area on each net (at this stage of connectivity) are computed as follows:

\[
\begin{align*}
\frac{\text{Area(Metal1a)}}{\text{Area(G1+G2)}} &= \frac{100}{5+5} = 10 \\
\frac{\text{Area(Metal1b)}}{\text{Area(G3+G4)}} &= \frac{200}{5+5} = 20 \\
\frac{\text{Area(Metal1c)}}{\text{Area(G5+G6)}} &= \frac{300}{5+5} = 30 \\
\frac{\text{Area(Metal1d)}}{\text{Area(G7+G8)}} &= \frac{400}{5+5} = 40 \\
\frac{\text{Area(Metal1e)}}{\text{Area(G9+G10)}} &= \frac{250}{2.5+2.5} = 50 \\
\end{align*}
\]

At this point the area ratios on the gates are

\[
\begin{align*}
\text{AC1:G1} &= \text{AC1:G2} = 10 \\
\text{AC1:G3} &= \text{AC1:G4} = 20 \\
\text{AC1:G5} &= \text{AC1:G6} = 30 \\
\text{AC1:G7} &= \text{AC1:G8} = 40 \\
\text{AC1:G9} &= \text{AC1:G10} = 50 \\
\end{align*}
\]

CONNECT Metal2 Metal1 By Via1

//***************************
//*** Connectivity zone 2 ***
//***************************

// AC2 contains all gate polygons, with the accumulated Metal/gate ratios from Metal1 and Metal2
// AC2 = NET AREA RATIO Metal2 Gate >= 0 ACCUMULATE AC1

// Write out accumulated ratios for debug
NET AREA RATIO PRINT AC2 NAR_AC2.txt

The ratios of Metal2 area to Gate area on each net (at this stage of connectivity) are computed as follows:

\[
\begin{align*}
\frac{\text{Area(Metal2a)}}{\text{Area(G1+G2+G3+G4)}} &= \frac{600}{5+5+5+5} = 30 \\
\frac{\text{Area(Metal2b)}}{\text{Area(G5+G6+G7+G8+G9+G10)}} &= \frac{1000}{5+5+5+2.5+2.5+2.5} = 40 \\
\end{align*}
\]

These area ratios are added to the existing area ratios on the gates. At this point the ratios on the gate are:

\[
\begin{align*}
\text{AC2:G1} &= \text{AC2:G2} = 30+10 = 40 \\
\text{AC2:G3} &= \text{AC2:G4} = 30+20 = 50 \\
\text{AC2:G5} &= \text{AC2:G6} = 40+30 = 70 \\
\text{AC2:G7} &= \text{AC2:G8} = 40+40 = 80 \\
\text{AC2:G9} &= \text{AC2:G10} = 40+50 = 90 \\
\end{align*}
\]

CONNECT Metal3 Metal2 By Via2

//***************************
//*** Connectivity zone 3 ***
//***************************

NAR.1 {

// AC3 contains those gate polygons whose accumulated Metal/Gate ratios from Metal1, Metal2, and Metal3 exceeds 150.
// AC3 = NET AREA RATIO Metal3 Gate > 150 ACCUMULATE AC2

}
COPY AC3

// Write out the accumulated ratios on the gate polygons in AC3
NET AREA RATIO PRINT AC3 NAR_AC3.txt

// Show the whole net as one error. Ornet performs an OR operation
// on all overlapping polygons on two input layers that are on the same net
Poly_ant = NET AREA RATIO Poly AC3 > 0
Metal1_ant = NET AREA RATIO Metal1 AC3 > 0
Metal2_ant = NET AREA RATIO Metal2 AC3 > 0
Metal3_ant = NET AREA RATIO Metal3 AC3 > 0
X = ORNET Poly_ant Metal1_ant
Y = ORNET Metal2_ant Metal3_ant
ORNET X Y

The ratios of Metal3 area to Gate area on each net (at this stage of connectivity) are computed as follows:

\[
\text{Area(Metal3a)/Area(G1+G2+G3+G4+G5+G6+G7+G8+G9+G10) = \frac{3600}{(5+5+5+5+5+5+5+5+2.5+2.5)} = 80}
\]

These area ratios are added to the existing area ratios on the gates. At this point the ratios on the gate are:

AC3:G1 = AC3:G2 = 80+40=120
AC3:G3 = AC3:G4 = 80+50=130
AC3:G5 = AC3:G6 = 80+70=150
AC3:G7 = AC3:G8 = 80+80=160
AC3:G9 = AC3:G10 = 80+90=170

The rule specifies that accumulated ratios > 150 are output, so G7, G8, G9, and G10 should be output.

See Also

“NET AREA RATIO PRINT” (page 564), “NET” (page 545), and “ATTACH” (page 540)
NET AREA RATIO PRINT

NET AREA RATIO PRINT layer filename

Description

Prints the input net area ratio accumulation information to the specified file.

Each line in the destination file corresponds to a polygon in the input layer, and consists of its lower left vertex coordinates followed by the antenna ratio associated with the polygon. These are specified in user units. Here is an example of the filename output:

189.625 406 23.7767
217.625 68 20.814

A NET AREA RATIO operation with ACCUMULATE option must be executed prior to printing. For example, it can be an operation in a DRC rule check that generates output:

ANT.Metal1 { X = NET AREA RATIO Metal1 Gate> 400 ACCUMULATE 
NET AREA RATIO PRINT X ANT_M1.txt 
}

Output from Net Area Ratio Print operations is different from Net Area Ratio RDB output. Net Area Ratio Print does not generate a DRC results style database, and it uses only net area ratio accumulation layers for input, and for output statistics. Also, Net Area Ratio RDB allows you to specify non-ACCUMULATE layers for output.

Parameters

layer
A required derived polygon layer that must be the output of a Net Area Ratio ACCUMULATE operation.

filename
A required filename for the output ASCII file. The layer must be specified before the filename to prevent ambiguity, but the filename parameter can contain environment variables.

See Also

“NET AREA RATIO” (page 556), “NET” (page 545)
NOT

[polygon-layer =] layer1 NOT layer2
[polygon-layer =] NOT layer1 layer2

Description

Calculates the region formed by layer1 minus layer2.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived polygon layer</td>
</tr>
<tr>
<td>layer2</td>
<td>A drawn or derived polygon layer</td>
</tr>
</tbody>
</table>
OFFGRID

OFFGRID layer1 {s | sx sy}

Description

Reports an error for every offgrid vertex of drawn polygons and wires on the specified layer and specified grid. This command can flag off-grid vertices of merged or derived layers, whereas the Flag Offgrid command only checks drawn layers. Original drawn layers are merged before the offgrid check is performed. Note, this operation can be time consuming if applied to large layers. The FLAG OFFGRID command, which does not merge layers, is a less time consuming approach.

Parameters

layer1 A drawn or derived polygon layer.
{s|sx sy} Positive integer(s) in database units, that specify the snap grid. The value s is applied in both x and y directions, while sx is applied in the x direction and sy in the y direction.

Example

The following checks for metal1 vertices that are not on a 0.01 micron grid (assuming 1000 database units per user-unit) :

Metal1_Offgrid {
    OFFGRID metal1 10
}

OR

\([\text{polygon-layer =}] \ OR \ layer1 \ constraint\)

**Description**

Calculates the region formed by the union of layer1 and layer2.

**Parameters**

- **layer1**: A drawn or derived polygon layer
- **constraint**: Constraint is any range defined by “Constraints” (page 491) where the constraints values must be integer.
ORNET

ORNET layer1 layer2 [by net | by shape]

Description

Note: This operation is performed flat in hierarchical applications; it has largely been replaced by “NET AREA RATIO” with the RDB option.

Performs a Boolean OR operation on all overlapping layer1 and layer2 polygons on the same net. Generates output equivalent to the polygon data on layer2 if layer1 is empty and vice versa.

Net Area, Net Area Ratio, and other Ornet operations having the same settings are the only operations that can have an input layer derived by Ornet because Ornet generates unmerged data. However, a layer derived by Ornet can be output to the DRC results database (which should be its primary use.) The connectivity on the input layers must be established.

Parameters

layer1 A drawn or derived layer.
layer2 A drawn or derived layer.
[BY net] An optional keyword that instructs L-Edit to merge overlapping layer1 and layer2 polygons that are on the same net. This is the default behavior if you do not include BY NET or BY SHAPE in the statement.
[BY shape] An optional keyword that instructs L-Edit not to merge any overlapping layer1 and layer2 polygons on the same net, but to output individual polygons instead.

See Also

“NET AREA RATIO” (page 556) and “NET AREA RATIO PRINT” (page 564)
OUTSIDE

[\text{polygon-layer} = \text{layer1} \ [\text{NOT}] \ \text{OUTSIDE} \ \text{layer2}]

[\text{polygon-layer} = \text{layer1} \ \text{layer2}]

\text{Description}

Produces all layer1 polygons that are completely outside layer2 polygons. Touching from the outside is considered to be outside. The \text{NOT} option produces those layer1 polygons not produced by the corresponding \text{OUTSIDE} operation.

\text{Parameters}

- \text{layer1} \quad \text{A drawn or derived polygon layer}
- \text{layer2} \quad \text{A drawn or derived polygon layer}
OUTSIDE EDGE

[edge-layer =] [NOT] OUTside EDGE layer1 layer2
[edge-layer =] layer1 [NOT] OUTside EDGE layer2

Description

Produces all layer1 edge segments that are completely outside layer2 polygons. The NOT option produces all layer1 edges not produced by the corresponding OUTSIDE EDGE operation.

Parameters

<table>
<thead>
<tr>
<th>layer1</th>
<th>A drawn or derived polygon layer or a derived edge layer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer2</td>
<td>A drawn or derived polygon layer.</td>
</tr>
</tbody>
</table>
PATH LENGTH

[\text{edge-layer} =] \text{PATH LENGTH layer constraint}

Description

Produces layer1 edges where the length of a contiguous set of edges conforms to the constraint.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A derived edge layer.</td>
</tr>
<tr>
<td>Constraint</td>
<td>Constraint is any range defined by “Constraints” (page 491) where the constraints values must be a non-negative real number in user units.</td>
</tr>
</tbody>
</table>
**PERIMETER**

```
[polygon-layer =] PERIMETER layer1 constraint
```

*Description*

Produces layer1 polygons whose perimeter conforms to the constraint.

*Parameters*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived polygon layer</td>
</tr>
<tr>
<td>constraint</td>
<td>Constraint is any range defined by “Constraints” (page 491) where the constraints values must be a non-negative real number.</td>
</tr>
</tbody>
</table>
POLYGON

POLYGON {x y} ... layer1

Description

Defines a polygon having the specified coordinates on the specified layer, placed in the toplevel cell of the design. Two (x,y) coordinate pairs are interpreted as an orthogonal rectangle. If more than two points are specified, the first and last point are connected, and do not need to be the same. The polygon can be specified in clockwise or counterclockwise order, but must be non self intersecting.

Parameters

- (x,y) A pair of real numbers in user units that specify the vertices of the polygon
- layer1 A drawn polygon layer previously defined with a LAYER statement.

Examples

| LAYER EXCL 64 |
| POLYGON -100 -220.5 100 220.5 EXCL |
POLYNET

POLYNET layer

Description

Creates a net for each layer polygon. Note that this operation is performed flat even in hierarchical tools.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer</td>
<td>A drawn or derived layer.</td>
</tr>
</tbody>
</table>

Note: Polynet is an obsolete Calibre command, but Tanner support is provided for compatibility with older versions of Calibre.


**PRECISION**

**PRECISION number**

*Description*

Specifies the ratio of user units to database internal units. This statement has the same meaning as “Database resolution” in L-Edit Setup > Design, and must have the same value. This command can appear only once in the command file. The default PRECISION is 1000 if this command is not present.

*Parameters*

**number**

A positive integer that defines the database resolution

*Examples*

```
PRECISION 1000  // 1000 database units per micron
```
**RECTANGLE**

```
RECTANGLE layer [constraint1 [BY constraint2]] [ASPECT constraint3]
[ORTHOGONAL ONLY | MEASURE EXTENTS]
```

**Description**

Produces rectangles from layer1. If constraints are specified, only rectangles whose dimensions conform to the constraint are produced. The NOT option produces layer1 rectangles not produced by the corresponding RECTANGLE operation.

**Parameters**

- **layer1**
  A drawn or derived polygon layer

- **constraint1**
  Constraint is any range defined by “Constraints” (page 491) where the constraints values must be a non-negative real number. If constraint1 is specified but constraint2 is not specified, then the rectangle is selected if either dimension of the rectangle satisfies the constraint.

- **BY constraint2**
  Constraint is any range defined by “Constraints” (page 491) where the constraints values must be a non-negative real number.

- **ASPECT constraint3**
  Specifies the ratio of the longer side to the shorter side that a rectangle must have in order to be output.

- **ORTHOGONAL ONLY**
  Restricts selection to rectangles with sides that are parallel to the horizontal and vertical coordinate axis.

- **MEASURE EXTENTS**
  Selects polygons based on their MBB. May not be specified with ORTHOGONAL ONLY.
RECTANGLE ENCLOSURE

[polygon-layer =] layer1 layer2 [intersection_filter] [OUTSIDE ALSO]
[ORTHOGONAL ONLY] {RECTANGLE _RULE [rectangle_rule...]}

Description

Rectangle enclosure uses a sequential process of elimination to make enclosure checks more efficient when multiple rules must be applied. Geometry is tested first to confirm rectangularity, then against the global parameters ABUT, SINGULAR or OUTSIDE ONLY (if used), then against each of eight rectangle rule relations (of each edge to each of four values, in both clockwise and counter-clockwise directions), which are of the type GOOD or BAD.

GOOD type rules output rectangles that fail any one of their criteria and do not output rectangles that satisfy all of their criteria. BAD type rules output rectangles that meet all of their criteria.

If the ABUT constraint is specified, intersecting edges will be measured. If the SINGULAR constraint is specified, point-to-point intersections, including self-intersections will also be measured. These options can be used together in any combination.

Parameters

layer1 A drawn or derived polygon layer
layer2 A drawn or derived polygon layer
intersection filter Optional parameters that permit intersecting edge or point pairs to be measured using the form

ABUT [constraint] [SINGULAR]

Used when a certain distance between intersecting objects is acceptable. You can specify any combination of these keywords in one operation.

ABUT [constraint]

Measures the separation between intersecting edges, if the angle between them conforms to the constraint, specified in degrees.

The default value is >= 0 < 180. Optionally, you can enter a non-negative real number less than 180. Single operator constraints such as < 90 and > 135 are interpreted as >= 0 < 90 and > 135 < 180.

Note that if the constraint modifier includes zero in its range (for example, < 90, == 0, >= 0 < 45), then any edges A of X and B of Y, which are coincident inside are also output because the angle between the exterior side of A and the interior side of B is zero. There is no measurement involved in this event, and polygon containment criteria are not applied.
Examples

For example, you would use Rectangle Enclosure instead of iterations of the Enclosure rule to test the following case:

All contacts must be enclosed by metal of .15. However, if two opposite sides are at least .5 then the two other sides can touch, or, if two opposite sides are at least .4 the two other sides can be as close as .05, or if two opposite sides are at least .3 the two other sides can be as close as .3. The contacts must be fully enclosed, and acute abutments are considered errors.

You would use the following rules:
RECTANGLE ENCLOSURE contact metal ABUT > 0 < 90 SINGULAR OUTSIDE ALSO

GOOD  0.00  0.50  0.00  0.50  
GOOD  0.05  0.40  0.05  0.40  
GOOD  0.10  0.30  0.10  0.30  
GOOD  0.15  0.15  0.15  0.15  

The tool checks each enclosure of contact by metal against the four spacing conditions listed, in the order the conditions appear. Any rectangle that does not satisfy at least one of the four conditions is classified as BAD and is output. Note that you do not need to explicitly specify any BAD conditions in order for a contact to be classified as BAD. A check having only BAD rules will output all layer1 rectangles. As such it is sufficient to specify all GOOD rules, unless there are only a few BAD conditions to check. BAD rules are best used to filter out gross violations.

Every edge of each rectangle is examined for each rule, first in clockwise order, and then again in counter-clockwise order. A relation satisfies a GOOD rule if all rules four elements of that relation are GOOD. A relation satisfies a BAD rule if all rules four elements of that relation are BAD. If the value 0 is used, the result is considered GOOD in a GOOD type rule and BAD in a BAD type rule.

Some possible results are shown below.
RECTANGLES

[polygon-layer =]  RECTANGLES width length {spacing | {width_spacing length_spacing} } [OFFSET {offset | {width_offset length_offset} } ] [ {INSIDE OF x1 y1 x2 y2} | {INSIDE OF LAYER layer} ] [MAINTAIN SPACING]

Description

Produces a layer consisting of a set of rectangles with specified width, spacing, and offset.

Parameters

**width length**

The width (x-axis) and length (y-axis) of the generated rectangles in user-units. Positive floating point numbers

**spacing**

The spacing between rectangles in user-units. A positive floating point number.

**width_spacing**

The x-axis and y-axis spacing, respectively, between rectangles in user-units. Positive floating point numbers.

**length_spacing**

**OFFSET**

A secondary keyword that specifies that rectangles are offset from the previous row/column of rectangles by a specified amount.

**offset**

Specifies the same offset in the x-axis and y-axis directions for each rectangle relative to the previous row/column of rectangles

**width_offset length_offset**

Specifies the offset in the x-axis and y-axis directions respectively for each rectangle relative to the previous row/column of rectangles. Positive floating point numbers.

**INSIDE OF x1 y1 x2 y2**

Specifies an area within the extent of the cell boundary to be filled with the specified rectangles. The parameters x1 y1 x2 y2 are four floating point numbers, in user-units, that indicate the lower-left (x1, y1) and upper-right (x2, y2) corners of the extent to be filled.

Negative numbers must be in parenthesis.

**INSIDE OF LAYER layer**

Specifies that rectangles are only produced within the boundary of the specified layer. The parameter layer is a string that specifies a derived or original polygon layer.

**MAINTAIN SPACING**

Specifies that no rectangle will be generated closer to another rectangle than the width_spacing and length_spacing parameters. This applies to clusters of rectangles between neighboring instances.
RESOLUTION

RESOLUTION \{ s \mid sx sy \}

Description

Specifies the grid size for offgrid checking by the FLAG OFFGRID command. This command can appear at most once in the command file.

Parameters

s A positive integer that specifies the x-direction and y-direction grid size for off-grid checking.

sx A positive integer that specifies the x-direction grid size for off-grid checking.

sy A positive integer that specifies the y-direction grid size for off-grid checking.

Examples

RESOLUTION 100 // define a one tenth micron grid for FLAG OFFGRID
SCONNECT

Syntax 1:
SCONNECT upper_layer lower_layer [... lower_layer] {BY contact_layer}[LINK name]

Syntax 2:
SCONNECT upper_layer lower_layer [LINK name] [ABUT ALSO]

Description

Establishes soft connections from the upper_layer polygons to lower_layer polygons through contact_layer. Connections are unidirectional; node numbers are passed from upper_layer to lower_layer, but not in the other direction.

Syntax 1:

Connections are made through the contact_layer argument if you use the BY keyword. Connectivity information is passed from upper_layer to lower_layer, through layerC objects, where lower_layer objects have positive overlap, with area common to both upper_layer and lower_layerC objects. Contact (layerC) polygons receive node numbers from upper_layer geometries. If more than one lower_layer is specified using Syntax 1, then shielding applies. The connection is made from the upper_layer to only the first lower_layer found at the particular location, in the specified order.

Syntax 2:

Connectivity information is passed from upper_layer to lower_layer, where upper_layer objects have positive overlap with lower_layer objects. The ABUT ALSO option allows connectivity information to be passed when upper_layer and lower_layer objects do not have positive overlap, but abut.

If two or more electrical nodes on upper_layer form soft connections to the same lower_layer polygon, then the electrical node that contains the largest number of polygon vertices (including contacts) is chosen.

Polygons on lower_layer that are not connected to any upper_layer in any Sconnect operation are called floating. Floating polygons receive unique node numbers if the secondary keyword LINK is not used, or if the secondary keyword LINK is used, but no node exists having the specified name. A warning is issued if the LINK keyword is used but no node exists with the specified name.

Parameters

**upper_layer**
A required original layer or layer set, or a derived polygon layer.

**lower_layer**
A required original layer or layer set, or a derived polygon layer. You can specify only one lower_layer in a Syntax 2 statement. Any layer specified as a lower_layer cannot simultaneously be a contact_layer in any Sconnect operation. Any lower_layer may be specified as a lower_layer in a different Sconnect operation.

**BY contact_layer**
A required secondary keyword set used with Syntax 1, where contact_layer is an original layer or layer set, or a derived polygon layer. The secondary keyword BY must always precede the name of this contact layer.
**LINK name**

An optional secondary keyword set, where name indicates an electrical node, that specifies the node number for floating polygons. Floating polygons are polygons on any specified lower_layer that are not connected to any upper_layer polygons. If a node with the name specified in the LINK option exists in the top level cell, then floating polygons receive this node name.

**ABUT ALSO**

A optional secondary keyword that specifies that abutment is considered to constitute overlap in this operation. Applies to Syntax 2 only.
SHRINK

[polygon-layer =] SHRINK layer1 [RIGHT BY value] [TOP BY value] [LEFT BY value][BOTTOM BY value]

Description

Performs inward translation of the input layer edges in the direction of the x-axis, y-axis, or both.

Parameters

layer1

An drawn or derived polygon layer or derived edge layer.

RIGHT BY value

Translate the right edge(s) of each polygon on layer1, toward the inside of each polygon layer1, by the specified value. Value is a non-negative real number in user units.

TOP BY value

Translate the top edge(s) of each polygon on layer1, toward the inside of each polygon layer1, by the specified value. Value is a non-negative real number in user units.

LEFT BY value

Translate the left edge(s) of each polygon on layer1, toward the inside of each polygon layer1, by the specified value. Value is a non-negative real number in user units.

BOTTOM BY value

Translate the bottom edge(s) of each polygon on layer1, toward the inside of each polygon layer1, by the specified value. Value is a non-negative real number in user units.
SIZE

[polygon-layer =] SIZE layer1 BY size_value [OVERLAP ONLY|INSIDE OF|OUTSIDE OF] layer2 [STEP step_value] [TRUNCATE distance]

[polygon-layer =] SIZE layer1 BY size_value [UNDEROVER|OVERUNDER] [TRUNCATE distance]

**Description**

Performs a size up or size down on a layer. A positive size_value performs a size up, or grow, and a negative size_value performs a size down, or shrink.

**Parameters**

- **layer1**: A drawn or derived polygon layer
- **size_value**: The amount to grow or shrink the layer1 polygons. This value must be positive if either UNDEROVER or OVERUNDER are specified.
- **UNDEROVER**: Undersize then oversize layer1 in a single operation.
- **OVERUNDER**: Oversize then undersize layer1 in a single operation.
- **OVERLAP ONLY**: Specifies that only the overlapping regions of the oversized polygons are output.
- **INSIDE OF layer2**: Constrains layer1 to travel inside layer2 when performing the SIZE operation. size_value must be positive when this option is used.
- **OUTSIDE OF layer2**: Constrains layer1 to travel outside layer2 when performing the SIZE operation. size_value must be positive when this option is used.
- **STEP step_value**: The step size of the sizing process. If size_value is not evenly divisible by step_value then the last step may be smaller than step_value in order to size by exactly size_value.
- **TRUNCATE distance**: The truncation distance to prevent large spikes when the size operation is performed on small acute angles. The default value is 1/COS(67.5), approximately 2.61. At the default truncation distance, corners at edges of \( \geq 45 \) degree angles will not be truncated, corners at edges of \(< 45 \) degrees will be truncated.

**Optimizing SIZE Performance**

The SIZE command, with STEP option as shown below:

```
SIZE layer1 BY size_value [INSIDE OF|OUTSIDE OF] layer2 [STEP step_value]
```

will perform a sequence of size operations by step_value, until a total sizing of size_value has been achieved (the last step_value may be reduced to obtain a size of exactly size_value). Certain foundry DRC file contain very small values of step_value, resulting in a very large number of steps and slow performance on these commands. Temporarily removing these commands by commenting them out can significantly improve performance of the whole DRC job. You will also need to comment out layers that derive from these operations.
SNAP

[polygon-layer =] SNAP layer1 {r | x y}

Description

Produces a layer by snapping the vertices of the input layer to the specified grid. 45-degree edges are preserved.

Parameters

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAYER1</td>
<td>A drawn or derived polygon layer.</td>
</tr>
<tr>
<td>r</td>
<td>X Y</td>
</tr>
</tbody>
</table>
STAMP

STAMP layer1 BY layer2 [ABUT ALSO]

Description

Produces a layer containing all layer1 polygons overlapped by layer2 polygons, and then transfers connectivity from the layer2 polygons onto the output layer polygons. If a layer1 polygon is overlapped by two or more layer2 polygons from different nets, or not overlapped at all, then the Stamp operation does not output the layer1 polygon. Warning messages report missing or conflicting connections. The ABUT ALSO option allows polygon abutment (touching at an edge) to establish a valid connection.

Parameters

layer1 A drawn or derived polygon layer.
layer2 A drawn or derived polygon layer.
ABUT ALSO Specifies that abutment of layer1 and layer2 is treated as an overlap.
SVRF ERROR

SVRF ERROR message

Description

This specification statement, where message is a string, generates a fatal rule file compiler error with message as the error message, if the statement is encountered during compilation. Since it always generates an error, it should appear in a conditional statement.

Parameters

message A text string to be displayed, typically as a message for a compilation error.

Example

```c
#ifndef ANTENNA_SET
SVRF ERROR "Do not run these rules unless ANTENNA is set"
#endif
```
TEXT DEPTH

TEXT DEPTH [PRIMARY | ALL | number]

Description

The TEXT DEPTH statement specifies the hierarchical depth for using text objects from the layout database for application in net naming. TEXT DEPTH ALL selects text objects from throughout the hierarchy, TEXT DEPTH PRIMARY selects only text objects from the top-level cell, and TEXT DEPTH n1 selects text objects n1 levels below the top-level cell. TEXT DEPTH PRIMARY is the default.

Text objects that come from lower levels of the hierarchy are transformed to the top-level coordinate space and are replicated according to the hierarchical structure of the design. Such text objects then behave as if they originated at the top level; this is true in flat as well as hierarchical applications. This statement supports connectivity extraction only. It does not influence text objects used by the With Text operation.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIMARY</td>
<td>Specifies that only text objects from the top-level cell are selected. This is the default behavior if you do not include this statement in the rule file.</td>
</tr>
<tr>
<td>ALL</td>
<td>Specifies that free-standing text objects from throughout the hierarchy are used as top-level text.</td>
</tr>
<tr>
<td>number</td>
<td>An non-negative integer that specifies to use text objects from number levels below the top-level cell. Specifying zero is equivalent to PRIMARY.</td>
</tr>
</tbody>
</table>
TEXT LAYER

TEXT LAYER layer [...]layer]

Description

Specifies the layers in the database from which text is used for net naming in connectivity extraction. Only text objects on layers that appear in Text Layer specification statements will be used in net naming. Thus, if there are no Text Layer specification statements in the rule file, then no database text objects are used by the connectivity extractor. This statement may be specified any number of times.

This statement affects connectivity extraction only; it does not influence text objects used by With Text or Expand Text operations in the rule file.

Parameters

layer

A layer name or number of a drawn layer from which to read text. You can specify layer any number of times in one statement. Layer sets are allowed and are equivalent to specifying each layer of the set individually.

Example

Ports on metal2 will not be used to label nets, because metal2 is not in the TEXT LAYER list.
TEXT LAYER poly metall diff 16
TITLE

TITLE rulesetname

Description

Specifies the title of the rule set. This command can appear only once in the command file.

Parameters

rulesetname A text string.

Examples

TITLE “DRC command file for 0.18 micron process”
TOUCH

[polygon-layer =] layer1 [NOT] TOUCH layer2 [constraint [BY NET]]
[polygon-layer =] [NOT] TOUCH layer1 layer2 [constraint [BY NET]]

Description

Produces all layer1 polygons that are completely outside layer2 polygons but share an edge with layer2 polygons. The NOT option produces those layer1 polygons not produced by the corresponding TOUCH operation. The input layers will be merged if the constraint option is used, and will run slower than the TOUCH operation with no constraint.

(This option is not yet supported.) If BY NET is specified, the constraint applies to the number of layer2 polygons on distinct nets that CUT layer1.

Parameters

layer1 A drawn or derived polygon layer
layer2 A drawn or derived polygon layer
constraint Specifies the number of layer2 polygons that a layer1 polygon must touch, in order to be selected by the TOUCH operation. Constraint is any range defined by “Constraints” (page 491) where the constraints values must be integer.

BY NET When the BY NET option is used, then a layer1 polygon is selected when the specified number of layer2 polygons on distinct nets touch the layer1 polygon. Layer2 must have connectivity.
TOUCH EDGE

```
[edge-layer =] [NOT] TOUCH EDGE layer1 layer2
[edge-layer =] layer1 [NOT] TOUCH EDGE layer2
```

**Description**

Produces all layer1 edges that touch layer2 edges. TOUCH EDGE produces the whole edge on layer1 when part of an edge is coincident, whereas COINCIDENT EDGE will just produce that segment that touches. The NOT option produces all layer1 edges not produced by the corresponding TOUCH EDGE operation.

**Parameters**

- **layer1**
  - A drawn or derived polygon layer or a derived edge layer.

- **layer2**
  - A drawn or derived polygon layer or a derived edge layer.
TOUCH INSIDE EDGE

[\text{edge-layer} =] [\text{NOT}] \text{ TOUCH INSide EDGE layer1 layer2}  
[\text{edge-layer} =] \text{layer1 [NOT] TOUCH INSide EDGE layer2}

\textbf{Description}

Produces all layer1 edges that touch layer2 edges from the inside. TOUCH INSIDE EDGE produces the whole edge on layer1 when part of an edge is inside coincident, whereas COINCIDENT INSIDE EDGE will just produce the segment that touches. The NOT option produces all layer1 edges not produced by the corresponding TOUCH INSIDE EDGE operation.

\textbf{Parameters}

\begin{itemize}
  \item \text{layer1} \hspace{1cm} \text{A drawn or derived polygon layer or a derived edge layer.}
  \item \text{layer2} \hspace{1cm} \text{A drawn or derived polygon layer or a derived edge layer.}
\end{itemize}
TOUCH OUTSIDE EDGE

[edge-layer =] [NOT] TOUCH OUTside EDGE layer1 layer2
[edge-layer =] layer1 [NOT] TOUCH OUTside EDGE layer2

Description

Produces all layer1 edges that touch layer2 edges from the outside. TOUCH OUTSIDE EDGE produces the whole edge when part of an edge is outside coincident, whereas COINCIDENT OUTSIDE EDGE will just produce that segment that touches. The NOT option produces all layer1 edges not produced by the corresponding TOUCH OUTSIDE EDGE operation.

Parameters

- **layer1**: A drawn or derived polygon layer or a derived edge layer.
- **layer2**: A drawn or derived polygon layer or a derived edge layer.
**VARIABLE**

VARIABLE name {value | ENVIRONMENT}

**Description**

Allows the use of variables in place of numeric constants in the file.

Variables can be placed inside of Rule Check comments and resolved to their value in the Verification Error Navigator by placing a carat (^) in front of the variable name. To display the (^) character, precede it with a backslash (\).

**Parameters**

- **name**: A string that specifies the name of a variable
- **value**: A string, which is a real number or an equation. The equation can include references to other variables
- **ENVIRONMENT**: A keyword that specifies that name is defined as an Environment variable in the operating system.

**Examples**

The following commands:

```plaintext
VARIABLE POLY_WIDTH 2
Rule_PO.W { @ Poly width must be ^POLY_WIDTH microns.
   INT poly < POLY_WIDTH
}
```

are equivalent writing:

```plaintext
Rule_PO.W { @ Poly width must be 2 microns.
   INT poly < 2
```
VERTEX

[polygon-layer =] VERTEX layer1 constraint

Description

Produces layer1 polygons whose vertex count conforms to the constraint.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived polygon layer</td>
</tr>
<tr>
<td>constraint</td>
<td>Constraint is any range defined by “Constraints” (page 491) where the constraints values must be integer.</td>
</tr>
</tbody>
</table>
VIRTUAL CONNECT COLON

VIRTUAL CONNECT COLON \{NO \mid YES\}

**Description**

Specifies that net names containing the colon character (:) are treated as virtually connected. Any set of geometrically disjoint nets with net labels that have the same name up to the first colon in the name are treated as a single net, if Virtual Connect Colon Yes is specified. The colon and all characters after it are ignored when net names are matched. Colons can appear anywhere in the name with the exception that a colon at the beginning of a name is treated as a regular character (that is, it has no special effect).

**Note:**

Note that you must also include a “VIRTUAL CONNECT NAME” command in the file for VIRTUAL CONNECT COLON YES to work.

**Parameters**

<table>
<thead>
<tr>
<th>NO</th>
<th>Specifies for the tool to not create virtual connections for net names containing a colon character. This is the default behavior when you do not include this statement in the rule file</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>Specifies for the tool to create virtual connections for net names containing a colon character.</td>
</tr>
</tbody>
</table>

**Examples**

VIRTUAL CONNECT COLON YES

1. Two geometrically distinct nets labeled N1: and N1: will be virtually connected. The name of the resulting net will be N1.

2. Two geometrically distinct nets labeled N1 and N1: are NOT virtually connected because the first N1 does not contain a colon. A warning is given about two nets with the same name and the name N1 is randomly assigned to one of them.

3. Geometrically distinct nets labeled VDD:, VDD:2, VDD:3, and will be virtually connected. The name given will be VDD.
VIRTUAL CONNECT NAME

VIRTUAL CONNECT NAME net_name [...] net_name

Description

Specifies virtual connections for the specified net names. Any set of geometrically disjoint nets that share the same name are treated as a single net, if that name appears in a Virtual Connect Name statement. This statement can be specified any number of times. Each name is a case-insensitive net name. If Virtual Connect Colon is also specified, then Virtual Connect Name operates on names after all colon suffixes have been stripped off.

Virtual connections are only made on the top-level cell. You can use the question mark (?) as a wildcard to match zero or more characters.

Parameters

net_name

A net name. You can specify net_name any number of times in one statement. The name can be a string variable.

Example

In the following example any set of geometrically distinct nets with the name VDD will become a single net, and any set of distinct nets with the name VCC will become a single net.

VIRTUAL CONNECT NAME "VDD" "VCC"
VIRTUAL CONNECT SEMICOLON AS COLON

Description

See “VIRTUAL CONNECT COLON” (page 552). Treats nets having a semicolon in their name as if the semicolon is a colon. VIRTUAL CONNECT COLON checks whether N is set to YES or NO. If VIRTUAL CONNECT SEMICOLON is set to YES then semicolons in net names are treated as colons.

Examples

[1] VIRTUAL CONNECT COLON YES
VIRTUAL CONNECT SEMICOLON AS COLON YES

Three net segments labeled XYZ:1 XYZ:2 XYZ:3 are virtually connected as a single net when a semicolon in a net label is treated as a colon.

[2] VIRTUAL CONNECT COLON NO
VIRTUAL CONNECT SEMICOLON AS COLON YES

or just

VIRTUAL CONNECT SEMICOLON AS COLON YES

No virtual connection will be made in three net segments labeled XYZ:1 XYZ:2 XYZ:3.
WITH EDGE

[polygon-layer =] [NOT] WITH EDGE layer1 layer2 [constraint]
[polygon-layer =] layer1 [NOT] WITH EDGE layer2 [constraint]

Description

Produces layer1 polygons that have edges or edge segments coincident with layer2. The constraint specifies the number of layer1 edges that a layer1 polygon must have on layer2 to be selected. The NOT option produces layer1 polygons that are not produced by the corresponding WITH EDGE operation.

Parameters

- **layer1**: A drawn or derived polygon layer
- **layer2**: A derived edge layer.
- **constraint**: Constraint is any range defined by “Constraints” (page 491) where the constraints values must be integer.
WITH NEIGHBOR

WITH NEIGHBOR layer1 [layer2] constraint1 constraint2 [SQUARE] [CENTERS]

Description

Selects rectangles that are orthogonal with respect to the database axes if they have the specified number of orthogonal rectangles within the specified distance.

Parameters

layer1  A drawn or derived polygon layer
layer2  A drawn or derived polygon layer
constraint1  Constraint is any range defined by “Constraints” (page 491) where the constraints values must be positive integers. Used to specify the number of orthogonal rectangles a given orthogonal rectangle must be in proximity of in order to be output.
SPACE constraint 2  Specifies the dimensions of the measurement region around a layer polygon. This is a required keyword and constraint containing non-negative, floating-point numbers in user units of length. The constraint may not be of the form “>”, “>=”, or “!=".
SQUARE  Specifies the measurement region formed by constraint2 using the SQUARE metric rather than the Euclidean metric.
CENTERS  Specifies the measurement region formed when constraint2 is measured from the center of a given rectangle rather than from its perimeter. Distances are measured to the centers of other rectangles rather than to their edges.
WITH TEXT

[NOT] WITH TEXT layer1 text_name [text_layer] [PRIMARY ONLY]

Description

Produces all layer1 polygons that intersect the location of text objects having the specified text_name. The command parameter order must be observed. If text_layer is specified, then only text objects with the specified name on layer text_layer are considered, otherwise all layers are considered.

The Text, Text Depth, and Text Layer statements have no effect on With Text operations.

Parameters

- **layer**: A drawn or derived polygon layer.
- **text_name**: The name of a text object, which can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. The name can be a string variable (see Variable).
- **text_layer**: A drawn layer containing text objects. Text_layer can be a layer name or GDS number.
- **PRIMARY ONLY**: Specifies that only top-level text is to be considered.

Example:

The text layer equal to 40 in the example below means to look only on layer 40 for “TEXTPAD”, “PROBEPAD”, and “SCRIBE”. If the “40” were not present, then look on all layers.

```
LAYER pad 40
pad_t   = pad WITH TEXT "TESTPAD"  40
pad_p   = pad WITH TEXT "PROBEPAD" 40
pad_sc  = pad WITH TEXT "SCRIBE"   40
```
WITH WIDTH

[polygon-layer =] WITH WIDTH layer1 constraint

Description

Selects those polygons or portions of polygons that satisfy the width constraint. For constraints >w and orthogonal geometry, this is equivalent to doing a SIZE layer1 BY (w/2) UNDEROVER.

Parameters

layer1 A drawn or derived polygon layer.
Constraint Specifies the width that the selected polygon or portion of a polygons must have in order to be produced. Constraints of type > or >= are allowed.
**XOR**

```
[polygon-layer =] layer1 XOR layer2
[polygon-layer =] XOR layer1 layer2
```

**Description**

Calculates the region formed by both layers minus the region shared by both layers.

**Parameters**

- **layer1**: A drawn or derived polygon layer
- **layer2**: A drawn or derived polygon layer
Supported Calibre Commands

AND (See “Unsupported Options”)
Angle
Area
Attach
Coincident Edge
Coincident Inside Edge
Coincident Outside Edge
Connect
Convex Edge
Copy
Cut (See “Unsupported Options”)
Density (See “Unsupported Options”)
Device
Disconnect
DMACRO and CMACRO
Donut
Drawn Acute
Drawn Offgrid
Drawn Skew
DRC Incremental Connect
DRC Maximum Results
DRC Print Area
DRC Print Perimeter
DRC Select Check
DRC Tolerance Factor
DRC Unselect Check
Enc
Enclose (See “Unsupported Options”)
Enclose Rectangle
Exclude Cell
Expand Edge
Expand Cell
Expand Text
Extent
Extents (See “Unsupported Options”)
External
Flag Acute
Flag Nonsimple
Flag Offgrid
FLAG
POLYGONVERTEXLIMIT
FLAG SKEW
FLAG WIREVERTEXLIMIT
FLAG ZEROWIDTH WIRES
Flatten
Flatten Cell
Flatten Inside Cell
Group
Grow
Holes
Include
Inside
Inside Cell
Inside Edge
Int
Interact (See “Unsupported Options”)
Label Order
Layer
Layer Map
Layer Resolution
Length
Merge
Net
Net Area
Net Area Ratio (See “Unsupported Options”)
Net Area Ratio Print
NOT
Not Angle
Not Area
Not Coincident Edge
Not Coincident Inside Edge
Not Coincident Outside Edge
Not Cut
Not Donut
Not Enclose
Not Enclose Rectangle
Not Inside
Not Inside Cell
Not Inside Edge
Not Interact
Not Length
Not Net
Not Outside
Not Outside Edge
Not Rectangle
Not Touch
Not Touch Edge
Not Touch Inside Edge
Not Touch Outside Edge
Not With Edge
Not With Neighbor
Not With Text
Not With Width
Offgrid
OR
Ornet
Outside
Outside Edge
Path Length
Perimeter
Polygon
Polynet
Port Depth
Port Layer Polygon
Port Layer Text
Push Cell
Precision
Rectangle
Rectangle Enclosure
Rectangles
Resolution
Sconnect
Shift
Shrink
Size
Snap
Stamp
SVRF Error
Text Depth
Text Layer
Title
Touch (See “Unsupported Options”)
Touch Edge (See “Unsupported Options”)

<table>
<thead>
<tr>
<th>Options</th>
<th>Virtual Connect Colon</th>
<th>Virtual Connect Depth</th>
<th>Virtual Connect Name</th>
<th>Virtual Connect Semicolon As Colon</th>
<th>With Edge</th>
<th>With Neighbor (See “Unsupported Options”)</th>
<th>With Text</th>
<th>With Width</th>
<th>XOR (See “Unsupported Options”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch Inside Edge (See “Unsupported Options”)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Touch Outside Edge (See “Unsupported Options”)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Vertex</td>
<td>Vertex</td>
<td>Vertex</td>
<td>Vertex</td>
<td>Vertex</td>
<td></td>
<td>Vertex</td>
<td>Vertex</td>
<td></td>
</tr>
<tr>
<td>Vertex</td>
<td>Vertex</td>
<td>Vertex</td>
<td>Vertex</td>
<td>Vertex</td>
<td>Vertex</td>
<td></td>
<td>Vertex</td>
<td>Vertex</td>
<td></td>
</tr>
</tbody>
</table>
Unsupported Options

The following commands are used in Calibre DRC or netlist extraction. These commands are supported in HiPer Verify, but not all options are supported, as listed below. HiPer Verify will issue a warning or error for options on this list.

<table>
<thead>
<tr>
<th>Calibre Command</th>
<th>Unsupported Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Single Layer Syntax</td>
</tr>
<tr>
<td>Cut</td>
<td>By Net, Even, Odd</td>
</tr>
<tr>
<td>Density</td>
<td>By Polygon, Centered, Corner, Gradient, RDB</td>
</tr>
<tr>
<td>Enclose</td>
<td>By Net, Even, Odd</td>
</tr>
<tr>
<td>Extents</td>
<td>Squares, Inside of Layer</td>
</tr>
<tr>
<td>Interact</td>
<td>By Net, Even, Odd, Singular</td>
</tr>
<tr>
<td>Net Area Ratio</td>
<td>Inside Of Layer</td>
</tr>
<tr>
<td>Touch</td>
<td>By Net, Even, Odd</td>
</tr>
<tr>
<td>Touch Edge</td>
<td>Endpoint</td>
</tr>
<tr>
<td>Touch Inside Edge</td>
<td>Endpoint</td>
</tr>
<tr>
<td>Touch Outside Edge</td>
<td>Endpoint</td>
</tr>
<tr>
<td>With Neighbor</td>
<td>Inside of Layer, [Not] Connected</td>
</tr>
<tr>
<td>XOR</td>
<td>Single Layer Syntax</td>
</tr>
</tbody>
</table>
## Unsupported Calibre DRC/Extraction Commands

The following commands are used in Calibre DRC or netlist extraction, but are not supported in L-Edit HiPer Verify.

- Capacitance Alias (no longer a Calibre command)
- Capacitance Ignore (no longer a Calibre command)
- Deangle
- Density Convolve
- Drawn Angled
- DRC Incremental Connect Warning
- DRC Select Check By Layer
- DRC Tolerance Factor NAR
- DRC Unselect Check By Layer
- Exclude Acute
- Exclude Angled
- Exclude Offgrid
- Exclude Skew
- Expand Cell Text
- Extent Cell
- Extent Drawn
- Flag Angled
- Flag Nonsimple Path
- Hcell
- Layout Allow Duplicate Cell
- Layout Bump2
- Layout Case
- Layout Cell List
- Layout Depth
- Layout Error On Input
- Layout Ignore Text
- Layout Input Exception Severity
- Layout Magnify
- Layout Merge On Input
- Layout Polygon
- Layout Preserve Case
- Layout Preserve Cell List
- Layout Property Audit
- Layout Property Text
- Layout Rename Cell
- Layout Rename Text
- Layout Text
- Layout Use Database Precision
- Layout Windel
- Layout Windel Cell
- Layout Windel Layer
- Layout Window
- Layout Window Cell
- Layout Window Clip
- Layout Window Layer
- Magnify
- Push
- Rotate
- Snap Offgrid
- SVRF Message
- TDDRC
- Text
Unsupported Database Commands

The following commands control the input and output of data in Calibre. They are not relevant in HiPer Verify. Though they are not supported, L-Edit recognizes these commands to prevent error messages, so HiPer Verify will not issue a warning for commands on this list.

- DRC Check Map
- DRC Keep Empty
- DRC Maximum Vertex
- DRC Results Database Libname
- DRC Results Database Precision
- DRC Summary Report
- Layer Directory
- Layout Path
- Layout Path2
- Layout Primary
- Layout Primary2
- Layout Process Box Record
- Layout System
- Layout System2
- Mask Results Database
- Mask SVDB Directory
Unsupported Database Classification Commands

The following commands are used by Calibre for recognition of certain design styles or for hierarchy recognition algorithms. They are not relevant in HiPer Verify.

Layout Base Cell
Layout Base Layer
Layout Top Layer
Unsupported Design for Manufacturing Commands

The following commands are used by the Calibre DFM applications. They are not relevant in HiPer Verify. HiPer Verify will not issue a warning for commands on this list.

- DFM Analyze
- DFM Create Layer
- DFM Critical Area
- DFM Database
- DFM Expand Edge
- DFM Expand Enclosure
- DFM Fill
- DFM Function
- DFM Grow
- DFM Histogram
- DFM Measure
- DFM Property
- DFM RDB
- DFM Select Check
- DFM Size
- DFM Spec Fill
- DFM Spec Fill Optimizer
- DFM Spec Fill Shape
- DFM Transition
## Unsupported Electrical Rule Check Commands

The following commands are used by Calibre for performing Electrical Rule Checking (ERC) during LVS. HiPer Verify does not support ERC. HiPer Verify will not issue a warning for commands on this list.

<table>
<thead>
<tr>
<th>ERC Cell Name</th>
<th>ERC Pathchk</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERC Check Text</td>
<td>ERC Results Database</td>
</tr>
<tr>
<td>ERC Keep Empty</td>
<td>ERC Select Check</td>
</tr>
<tr>
<td>ERC Maximum Results</td>
<td>ERC Summary Report</td>
</tr>
<tr>
<td>ERC Maximum Vertex</td>
<td>ERC Unselect Check</td>
</tr>
<tr>
<td>ERC Path Also</td>
<td>Pathchk</td>
</tr>
</tbody>
</table>
Unsupported Inductance Commands

The following commands are used by the Calibre parasitic inductance extraction application, and are not relevant in HiPer Verify.

Inductance MiCheck
Inductance Wire
Unit Inductance
## Unsupported Litho Commands

The following commands are used by the Calibre OPC, RET, FRACTURE, or ORC applications and are not relevant in HiPer Verify.

<table>
<thead>
<tr>
<th>Command</th>
<th>Command</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBCLASSIFY</td>
<td>LithoOPCverify</td>
<td>Opcbias</td>
</tr>
<tr>
<td>LithoDenseOPC</td>
<td>LithoORC</td>
<td>Opclineend</td>
</tr>
<tr>
<td>Litho File</td>
<td>LithoPrintimage</td>
<td>Opcsbar</td>
</tr>
<tr>
<td>LithoOPC</td>
<td>LithoPSMgate</td>
<td></td>
</tr>
</tbody>
</table>
Unsupported LVS Commands

The following commands are used by Calibre for performing LVS. Calibre performs netlist extraction and comparison in a single step, whereas HiPer Verify performs netlist extraction, and then LVS is a separate application. These are not used in either HiPer Verify or in LVS. HiPer Verify will not issue a warning for commands on this list. There are almost equivalent or similar functionality for the italicized commands in LVS program or during netlist extraction.

Device Layer
LVSD Abort On ERC Error
LVSD Abort On Softchk
LVSD Abort On Supply Error
LVSD All Capacitor Pins Swappable
LVSD Box
LVSD Builtin Device Pin Swap
LVSD Cell List
LVSD Cell Supply
LVSD Check Port Names
LVSD Compare Case
LVSD Component Subtype Property
LVSD Component Type Property
LVSD Cpoint
LVSD Device Type
LVSD Discard Pins By Device
LVSD Downcase Device
LVSD Exclude Hcell
LVSD Execute ERC
LVSD Expand Seed Promotions
LVSD Expand Unbalanced Cells
LVSD Filter
LVSD Filter Unused Bipolar
LVSD Filter Unused Capacitors
LVSD Filter Unused Diodes
LVSD Filter Unused MOS
LVSD Filter Unused Option
LVSD Filter Unused Resistors
LVSD Global Layout Name
LVSD Globals Are Ports
LVSD Ground Name
LVSD Heap Directory
LVSD Ignore Ports
LVSD Ignore Trivial Named Ports
LVSD Inject Logic
LVSD Isolate Shorts
LVSD Map Device
LVSD Netlist All Texted Pins
LVSD Netlist Allow Inconsistent Model
LVSD Netlist Box Contents
LVSD Netlist Comment Coded Substrate
LVSD Netlist Unnamed Box Pins
LVSD NL Pin Locations
LVSD Non User Name
LVSD Out Of Range Exclude Zero
LVSD Pin Name Property
LVSD Power Name
LVSD Precise Interaction
LVSD Preserve Floating Top Nets
LVSD Preserve Parameterized Cells
LVSD Property Initialize
LVSD Property Map
LVSD Property Resolution Maximum
LVSD Push Devices
LVSD Recognize Gates
LVSD Recognize Gates Tolerance
LVSD Reduce
LVSD Reduce Parallel Bipolar
LVSD Reduce Parallel Capacitors
LVSD Reduce Parallel Diodes
LVSD Reduce Parallel MOS
LVSD Reduce Parallel Resistors
LVSD Reduce Semi Series MOS
LVSD Reduce Series Capacitors
LVSD Reduce Series MOS
LVSD Reduce Series Resistors
LVSD Reduce Split Gates
LVSD Reduction Priority
LVSD Report
LVSD Report Maximum
LVSD Report Option
LVSD Report Units
LVSD Report Warnings Top Only
LVSD Reverse WL
LVSD Show Seed Promotions
LVSD Show Seed Promotions Maximum
LVSD Signature Maximum
LVSD Soft Substrate Pins
LVSD Softchk
LVSD Spice Allow Duplicate Subcircuit Names
LVSD Spice Allow Floating Pins
LVSD Spice Allow Inline Parameters
LVSD Spice Allow Unquoted Strings
LVSD Spice Conditional LDD
LVSD Spice Cull Primitive Subcircuits
LVSD Spice Implied MOS Area
LVSD Spice Multiplier Name
LVSD Spice Option
LVSD Spice Override Globals
LVSD Spice Prefer Pins
LVSD Spice Redefine Param
LVSD Spice Rename Parameter
LVSD Spice Replicate Devices
LVSD Spice Slash Is Space
LVSD Spice Strict WL
LVSD Split Gate Ratio
LVSD Strict Subtypes
LVSD Write Injected Layout Netlist
LVSD Write Injected Source Netlist
LVSD Write Layout Netlist
LVSD Write Source Netlist
LVSD Source Case
LVSD Source Path
LVSD Source Primary
LVSD Source System
LVSD Trace Property
LVSD Unit Capacitance
LVSD Unit Length
LVSD Unit Resistance
## Unsupported Mask Data Prep Commands

The following commands are used for Mask Data Preparation, and are not relevant in Hiper Verify.

| Fracture HITACHI | Fracture NUFLARE | MDPmerge |
| Fracture JEOL    | Fracture VBOASIS | MDPstat   |
| Fracture MEBES   | MDP Embed        | MDPverify |
| Fracture MICRONIC| MDP Mapsize      |          |
Unsupported Parasitic Extraction Commands

The following commands are used by the Calibre parasitic extraction application to extract parasitic devices. They are not relevant in HiPer Verify. A similar capability is available in HiPer PX.

- Capacitance Order
- Parasitic Variation
- PEX BA Mapfile
- PEX Contact Capacitance
- PEX Density Estimate
- PEX Density Window
- PEX Driver File
- PEX Elayer
- PEX Exclude Distributed
- PEX Exclude Lumped
- PEX Fill Handling
- PEX Generate Driver_File Tag
- PEX Ground
- PEX Ground Layer
- PEX Ideal Xcell
- PEX Include Distributed
- PEX Include Lumped
- PEX Inductance Default PI
- PEX Inductance Differential Pair
- PEX Inductance Doprocess
- PEX Inductance Filter
- PEX Inductance Frequency
- PEX Inductance Parameters
- PEX Inductance Range
- PEX Inductance Returnpath
- PEX Inductance Self
- PEX Inductance Switch_Time
- PEX Inductance Victim
- PEX Inductance Victim_File
- PEX Inductance Victim_Path
- PEX InductanceFrequency
- PEX InductanceSwitch_Time
- PEX Magnify
- PEX Netlist ADMS
- PEX Netlist Character Map
- PEX Netlist Distributed
- PEX Netlist Lumped
- PEX Netlist Position File
- PEX Netlist Simple
- PEX Pin Order
- PEX Power
- PEX Probe File
- PEX Profile
- PEX Reduce CC
- PEX Reduce Coupled
- PEX Reduce Distributed
- PEX Reduce Lumped C
- PEX Reduce ROnly
- PEX Report Distributed
- PEX Report Lumped
- PEX Report Netsummary
- PEX Resistance Parameters
- PEX Skin Include
- PEX Temperature
- PEX Thickness EQN
- PEX Thickness Nominal
- PEX Threshold
- PEX Tolerance Distributed
- PEX Via Reduction Resistance
- Resistance Connection
- Resistance Device_Seed
- Resistance Rho
- Resistance Sheet
- Unit Time
21 HiPer Verify: Dracula Command Files

Introduction

This section provides a reference to Dracula® compatible DRC and Extract command file format.

Structure of a Dracula File

A Dracula format command file has the following major sections. Each block begins with a statement that identifies the block and ends with an *END statement. The *END statement is always the last line in the block. The block statements begin with an asterisk (*) in the first column.

“Description Block Commands” on page 727

The Description block contains Environment Setup commands and Geometry Flag commands.

The Description block describes the system and contains information about the circuit to be checked, such as input/output devices, file names, and formats.

“Input-Layer Block Commands” on page 736

The Input-Layer block contains Drawn Layer and Text Layer definition commands.

“Operation Block Commands” on page 744

The Operation block contains the Boolean, size, selection, area, rule check and remaining commands.

Create as many operation blocks as necessary to define your rules and processes. Individual definitions are automatically concatenated in the order you specify.

Command Usage

Commands can be used as layer derivation statements, or in rule check statements.

- A layer derivation statement consists of directing the results of a command to a named layer.
  
  AND POLY ACTIVE GATE

- A rule check statement uses the OUTPUT keyword to direct the results of a command to the Verification Error Navigator. The name after the OUTPUT keyword is the rulename that will appear in the Verification Error Navigator. The number following the rulename is a layer number used by Dracula.

  EXT GATE LT 2.0 OUTPUT GAS1 64

- Any command, not only dimensional check operations, may direct errors to the Verification Error Navigator. For example

  * Dracula is a registered trademark of Cadence Design Systems, Inc.
SEL VIA OUTSIDE METAL V1 OUTPUT V1 64

Conjunctive Rules

Errors from dimensional check operations can be reprocessed using conjunctive rule syntax. The following commands can be used to create conjunctive rules:

- ENC
- EXT
- INT
- WIDTH
- LENGTH

Consider the following sequence for finding M1 segments that are narrower than 3.0, closer than 5.0 to other narrow segments, and longer than 10.0:

```
WIDTH M1 LT 3.0 &
EXT M1 LT 5.0 &
LENGTH M1 GE 10.0 OUTPUT M1_THINCLOSELONG 64
```

The & operator causes the results of the command to be placed on a temporary layer conjoined to M1. Any references to M1 after the & and before the next OUTPUT keyword will use the temporary layer, and will continue to filter the results until they are output by the OUTPUT statement. After the OUTPUT statement, usage of M1 will refer to the original M1 layer.

Conjunctive rename allows renaming of the result of a conductive rule. Consider the following sequence for finding M1 segments that are closer than 5.0 to other M1 segments, and longer than 10.0. The results of the first operation are put on a layer called &M1CLOSE.

```
EXT M1&M1CLOSE LT 5.0 &
LENGTH &M1CLOSE GE 10.0 OUTPUT M1_CLOSELONG 64
```

Dimensional Check statements can also output results to a named layer, for subsequent processing using the R or R’ flag. Consider the following sequence for finding GATE lengths less than 0.5:

```
AND POLY ACTIVE GATE
ENC [TR] GATE ACTIVE LT 0.001 GL
PLENGTH GL LT 0.5 OUTPUT GATELENGTH 64
```

Navigation in the Text Editor

To speed navigation in the text editor, you can right-click on a layer name to access the command “Go To Layer Definition” in the context-sensitive menu, which jumps the cursor to the line with the definition for that layer.
Attaching Text

In L-Edit, you do not directly attach text strings to layout geometries. Text strings are part of a separate layer and do not have pointers back to polygons. Dracula attaches text to its corresponding geometry according to your definition in the Input-Layer block.

The ATTACH command attaches text to a geometry on a layer. The following command will attach text on layer 9 to layer Metal. Text on layer 9 can now only be used to attach to metal, and will not attach to any other layer.

```* INPUT-LAYER
Metal = 9 TEXT = 9 ATTACH Metal```

The following command will attach text on layer 109 to layer Metal.

```* INPUT-LAYER
Metal = 9 TEXT = 109 ATTACH Metal```

The ATTACH command can also be written on a separate line than the layer definition:

```* INPUT-LAYER
Metal = 9
TEXT = 109 ATTACH Metal```

Text that is not attached to a specific layer can attach to any layer, as specified by the CONNECT-LAYER command, and the optional TEXTSEQUENCE COMMAND. In the following example, text on layer 9 will label any layer, according to CONNECT-LAYER and TEXTSEQUENCE, as there is no ATTACH command.

```* INPUT-LAYER
Metal = 9 TEXT = 9```

**Example**

```*INPUT-LAYER
diffus = 2
poly = 8
metal = 9 TEXT = 60
text = 20 ATTACH metal
CONNECT-LAYER = diffus poly metal
PAD-LAYER = vapox
*END```

In this example, the text on layer 60 is not specifically attached to one layer, so it attaches to one of the connect layers. The text on layer 20 can attach only to layer metal.

Dracula attaches text in the reverse order of the CONNECT-LAYER, where the metal layer has the highest priority and the diffusion layer the lowest. In the previous example, text strings with coordinates within the metal geometries are attached to the metal layer and eliminated from further attachments. All remaining text in the polysilicon is attached. Finally, the text in the diffusion is attached and Dracula discards any text outside the three layers.

**Net Naming Rules and Conventions**
Nets are formed in Dracula by the command

```
CONNECT layer1 layer2 BY connect-layer
```

Nets may be named by placing text labels (L-Edit ports) overlapping the polygons that form a net, thus assigning the net a name equal to the text of the label. The rules describing how labels are used to name nets in Dracula are described in this section. Note that SELECT LABEL, which selects polygons on a layer with a specified net name, is the only Dracula DRC command that specifically uses a net name. (This is distinct from the SELECT BY LABEL command, which selects polygons on a layer based on overlapping text labels, and is completely unrelated to nets.)

The CONNECT-LAYER command in the Input-Layer Block defines the conductor layers of the IC process from bottom to top. If you use a CONNECT command in the Operation block, you must use the CONNECT-LAYER command in the Input-Layer block. Specify only the layers that appear in the CONNECT commands (excluding the contact layers). You can specify multiple layer names on one line or use multiple lines.

The TEXTSEQUENCE command in the Input-Layer Block redefines the text sequence generated by the CONNECT-LAYER command. The TEXTSEQUENCE command reorders the interconnect layers for attaching text. Text attaches first to the rightmost argument, then in order from right to left, similar to the CONNECT-LAYER command.

**Translation to Calibre:**

```
CONNECT-LAYER = bottom-layer ... middle-layer ... top-layer
```

Translates to:

```
TEXT LAYER bottom-layer ... middle-layer ... top-layer
LABEL ORDER top-layer ... middle-layer ... bottom-layer
```

Note that layer order is reversed.

If TEXTSEQUENCE is present, then

```
CONNECT-LAYER = bottom-layer ... middle-layer ... top-layer
TEXTSEQUENCE = middle-layer ... bottom-layer
```

Translates to:

```
TEXT LAYER bottom-layer ... middle-layer ... top-layer
LABEL ORDER bottom-layer ... middle-layer ...
```

Note that layer order is reversed and CONNECT-LAYER is required, regardless of the presence of TEXTSEQUENCE.

**Simple Example of a Dracula File**

```
*DESCRIPTION
; Place environment commands in the DESCRIPTION block.
SCALE = .001 MICRON
RESOLUTION = .001 MICRON
```
FLAGNON45 = YES
FLAG-OFFGRID = YES
*END

*INPUT-LAYER
; Define input layers in the INPUT-LAYER block.
  POLY = 1
*END

*OPERATION
; Place Boolean, Select, Size, and Dimensional check operations
; in the OPTATION block
  WIDTH POLY LT 0.18 OUTPUT POW1A 63
*END

Description Block Commands
DELCEL

\[ \text{DELCEL} = \text{name ...} \]

**Description**

Excludes instances of specified cells from layout when processing DRC commands. Cells may also be excluded from DRC processing by checking “Exclude instances of this cell from DRC” in the Cell > Info dialog in L-Edit.

**Arguments**

- **name**: The name of a cell. If a cell name has spaces, then the name is in quotes. Name can be specified any number of times in one statement.

**Examples**

```
DELCEL LOGO PICTURE
```

**Calibre Format**

```
EXCLUDE CELL name ...
```
FLAG-ACUTEANGLE

FLAG-ACUTEANGLE = NO | YES

**Description**

Reports an error for any two consecutive edges of a drawn polygon or wire that form an acute angle. This command can appear only once in the command file.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Default. Do not report acute angles.</td>
</tr>
<tr>
<td>YES</td>
<td>Report acute angles.</td>
</tr>
</tbody>
</table>

**Examples**

FLAG-ACUTE = YES

**Calibre Format**

FLAG ACUTE {NO | YES}
FLAG-NON45

FLAG-NON45 = NO | YES

Description

Reports an error for any non-90 or non-45 degree edge of a drawn polygon or centerline segment of a drawn wire. Reporting is based on the angle of the edge in its cell coordinate space, not in coordinate space of instances of that cell. This command can appear only once in the command file.

Arguments

NO Default. Do not report non-90/non-45 degree edges.
YES Report non-90/non-45 degree edges.

Examples

FLAG-NON45 = YES

Calibre Format

FLAG SKEW {NO | YES}
FLAG-OFFGRID/FLAG-PTH-OFFGRID

FLAG-OFFGRID = NO | YES \{grid-value\}  
FLAG-PTH-OFFGRID = NO | YES

Description

Reports an error for every offgrid vertex of drawn polygons and wires. Also reports errors for offgrid instance placements, rotated instances, and instance arrays and scaling that could result in offgrid geometry. Checking and reporting is done in the context of the cell. The grid is defined by the RESOLUTION command, or can be overridden by the FLAG-OFFGRID command. This command can appear only once in the command file.

Note: L-Edit/DRC does not have separate control of offgrid checking for polygons and wires. If either of these options are YES, then L-Edit/DRC will check polygons, wires and instances for offgrid.

Arguments

NO Default. Do not report offgrid vertices and instances.
YES Report offgrid vertices and instances.
GRID-VALUE Grid value that overrides the RESOLUTION value.

Examples

FLAG-OFFGRID = YES

Calibre Format

FLAG OFFGRID YES
FLAG-SELFINTERS/FLAG-SELFTOUCH

FLAG-SELFINTERS = YES (FULL)

Description

Report an error if two edges on the same drawn polygon intersect or if filled region of the polygon is ambiguous. Wires are checked for intersections based on the outer boundary of the wire.

Self-intersecting polygons and wires are always reported as errors. Self intersecting polygons and wires are ignored by all layer generation and rule checking commands. A no argument in this command will be ignored, and self-intersections will be checked anyway. The FULL option tells DRACULA to flag re-entrant polygons. In L-Edit DRC these are always flagged, so the FULL option is not required. This command can appear only once in the command file.

Examples

FLAG-SELFINTERS = YES

Calibre Format

FLAG NONSIMPLE YES
RESOLUTION

RESOLUTION = step-size units

Description

Defines the grid size for offgrid checking by the FLAG-OFFGRID command. This command can appear only once in the Description Block.

Note: L-Edit/DRC will not automatically snap offgrid vertices to the RESOLUTION grid, as Dracula will.

Arguments

- **step-size**: A positive integer that defines the grid size for offgrid checking by the FLAG-OFFGRID command.
- **units**: MICRONS or MILS (MICRON, MIC, and MIL are also allowed.)

Examples

RESOLUTION = 0.010 MICRONS

Calibre Format

RESOLUTION 1/step-size
SCALE

SCALE = unit-size units

Description

Specifies the number of internal database units that equals one layout unit. This command can appear only once in the Description Block.

Arguments

unit-size A positive integer.
Units MICRONs or MILS (MICRON, MIC, and MIL are also allowed.)

Examples

scale= .001 MICRONs

Calibre Format

PRECISION 1/unit-size
TEXT-LEVEL

\[ \text{TEXT-LEVEL} = n_1:n_2 / n \]

**Description**

Specifies the number of levels of hierarchy from which to read text. This command applies to cell and composite texts. Count the depth levels from the primary cell or level 0.

**Note:**

TEXT-LEVEL = 0 has the same effect as the TEXT-PRI-ONLY = YES command.

**Arguments**

- \( n_1:n_2 \)  
  The range of levels of hierarchy from which Dracula reads text.

- \( n \)  
  The range from 0 to \( n \) levels of hierarchy from which text is read.

**Example**

In this example, Dracula reads text from the primary cell:

\[ \text{TEXT-LEVEL} = 0 \]

In this example, Dracula reads text from level 0 to level 3:

\[ \text{TEXT-LEVEL} = 3 \]

In this example, Dracula reads text from level 3 to level 5:

\[ \text{TEXT-LEVEL} = 3:5 \]

In this example, Dracula reads text from level 3 only:

\[ \text{TEXT-LEVEL} = 3:3 \]

**Translation to Calibre**

\[ \text{TEXT-LEVEL} = n \]

translates to: \( \text{TEXT DEPTH} = n \)

\[ \text{TEXT-LEVEL} = n_1:n_2 \]

translates to: \( \text{TEXT DEPTH} = >= n_1 <= n_2 \)  
(Not standard Calibre.)
Chapter 21: HiPer Verify: Dracula Command Files

TEXT-PRI-ONLY

TEXT-PRI-ONLY = YES/NO

Description

Processes only the text associated with the top-level (primary) cell in the layout. This command ignores text associated with cells nested below the top-level cell, even if the text is on the same text layer number. If you do not specify TEXT-PRI-ONLY, Dracula places all text at the top level and processes it. However, the TEXT-LEVEL command will override TEXT-PRI-ONLY.

Arguments

YES Translates to TEXT-DEPTH = PRIMARY.
NO Translates to TEXT-DEPTH = ALL (default).

Translation to Calibre

TEXT-PRI-ONLY = YES
translates to TEXT DEPTH = PRIMARY

TEXT-PRI-ONLY = NO
translates to TEXT DEPTH = ALL (default)

Note: Dracula default is to process text from all cells. Calibre default is to process text from toplevel (primary) cell only.

Input-Layer Block Commands
Layer Assignment

layer-name = layer-number {OFFGRID = off-grid} {DATATYPE = data-type} {TEXT = text-layer} {TEXTTYPE text-type} {ATTACH layer-name1}

TEXT = text-layer {TEXTTYPE text-type} {ATTACH layer-name1}

Description

Defines the name of a drawn layer in terms of its GDSII number.

A layer assignment statement is required in the command file in order to use a drawn layer in a layer derivation or DRC command when running DRC from within the L-Edit environment. Layer assignments statements can be used to map a layer name in the L-Edit editing environment to a different name in the DRC command file by assigning the same GDSII number to the layer in each location.

To make a layer in the tdb file to be equal to different layer name in the DRC command file then assign the same GDS number to that layer in Setup Layers > General in the tdb file, and to the new name in a layer assignment statement in the command file. A summary of mappings will be reported in the DRC Summary Report for all layer names that are different in the tdb file and command file.

Arguments

layer-name User-defined layer name. You can use the same layer name on more than one line to group layers from different layer numbers.

layer-number The GDS number of the assigned layer in the layout CAD system. Also, you can use inclusive range values (for example, metal = 1:3.

DATATYPE = data-type Specifies a datatype or range of datatypes for the corresponding GDS number, to assign to the layer. A range of datatypes is specified as DATATYPE=5:10. The default is all datatypes.

TEXT = text-layer Specifies the GDS layer number for text.

TEXTTYPE = text-type The GDS datatype or range of datatypes for the corresponding GDS number, to assign to the layer. Text-type is the same as datatype, but refers to text objects.

ATTACH layer-name1 The attach operation assigns names to extracted nets using text objects (ports) placed on the layout. It attaches the name of a text object on layer1 to a net containing a polygon on layer2 if the polygon completely covers the text object. The layer-name1 parameter is the layer name given to the text defined by the text-layer parameter. The layers specified by layer-name1 must appear in a CONNECT-LAYER or TEXTSEQUENCE command.

Unsupported Arguments

ANGLE = ALL/90/45/NON-45/ACUTE/NON-90/NON-ACUTE
Examples

Example 1:

Poly1 = 46

Example 2: Layer OD consists of GDS layers 3, 11, and 12.

OD = 3
OD = 11
OD = 12

Example 3: Using DATATYPE

METAL1-TIGHT = 49  DATATYPE = 1
METAL1 = 49 DATATYPE  0:3

METAL1 does not contain objects from layer 49 datatype 1, as these were used by METAL1-TIGHT.

Calibre Format

// Dracula command: Layer-name = layer-number

LAYER layer-name layer-number

// Dracula command: layer-name = layer-number DATATYPE = 5:10
LAYER layer-name 1000  // 1000 is a temporary GDS number
LAYER MAP  layer-number DATATYPE >=5 <= 10  1000
Layer-Name Definition

layer-name = layer-number {OFFGRID = off-grid} {DATATYPE = data-type} {TEXT = text-layer} {TEXTTYPE text-type} {ATTACH layer-name1} {ANGLE = ALL/90/45/NON-45/ACUTE/NON-90/NON-ACUTE}

TEXT = text-layer layer-purpose {TEXTTYPE text-type} {ATTACH layer-name1} {ANGLE = ALL/90/45/NON-45/ACUTE/NON-90/NON-ACUTE}

Description

The layer-name variable assigns names to layout layers. You can name layers from GDSII, Edge, EDIF, or CIF formats.

Note:

Dracula can read the same layer for both TEXT and CTEXT text, but you cannot attach text in the same layer to different layers.

For example, in the following some of the text attached to MET is lost:

POLY=3 CTEXT=29 ATTACH=POLY
MET=6 CTEXT=29 ATTACH=MET
VAPOX=7 TEXT=29

Arguments

layer-name User-defined layer name.
layer-number The GDS number for the layer name.
off-grid Translate to Calibre: layer-name_offgrid {OFFGRID layer-name off-grid}
data-type Specifies a datatype or range of datatypes for the corresponding GDS number, to assign to the layer. A range of datatypes is specified as DATATYPE=5:10. The default is all datatypes.
text-layer The text that is plotted when you specify the PLOT commands. Text can be designated as a layer-name so that it is associated with a layer number, for example, text = 60 ATTACH metal. Range value as entered with a colon, for example, TEXT=13:44. The default is that TEXT is not used.
text-type The GDS datatype or range of datatypes for the corresponding GDS number, to assign to the layer. Text-type is the same as datatype, but refers to text objects.
ATTACH layer-name1  The attach operation assigns names to extracted nets using text objects (ports) placed on the layout. It attaches the name of a text object on layer1 to a net containing a polygon on layer2 if the polygon completely covers the text object.

The layer-name1 parameter is the layer name given to the text defined by the text-layer parameter. The layers specified by layer-name1 must appear in a CONNECT-LAYER or TEXTSEQUENCE command.

**Calibre Format**

Example 1:

layer-name = layer-number DATATYPE datatype TEXT = text-layer TEXTTYPE text-type ATTACH layer-name1

This is the same as if the line had been separated into two lines, as shown below:

layer-name = layer-number DATATYPE datatype
TEXT = text-layer TEXTTYPE text-type ATTACH layer-name1

Example 2:

a)

**TEXT = gds-text-layer  ATTACH layer-name1**

Translation:

**TEXT LAYER gds-text-layer**
**ATTACH gds-text-layer layer-name1**

b)

**TEXT = 49 ATTACH Metal1**

Translation:

**TEXT LAYER 49**
**ATTACH 49 Metal1**

Example 3:

a)

**TEXT = gds-text-layer TEXTTYPE text-type ATTACH layer-name1**

Translation:

**LAYER TEMP 1001**
**LAYERMAP == gds-text-layer TEXTTYPE constraint text-type 1001**
b)

TEXT = 49 TEXTTYPE 2 ATTACH Metall

Translation:

LAYER TEMP 1001
LAYER MAP == 49 TEXTTYPE == 2 1001
TEXT LAYER TEMP
ATTACH TEMP Metall
CONNECT LAYER

CONNECT-LAYER = layer1 layer2 ...

Description

CONNECT-LAYER defines the layer sequence in which polygons on a net are examined for intersection and attachment with a net naming label. The process of assigning label names to nets gives first priority to explicit ATTACH commands, then to implicit attachment (labels on same layer as overlapping polygon), and then to the CONNECT-LAYER for the remaining text labels. When multiple polygons on different nets overlap a label, then the net with the polygon whose layer appears last in the CONNECT-LAYER list is labeled with the value of the net name. The TEXTSEQUENCE command overrides the sequence given by CONNECT-LAYER.

If you use a CONNECT command in the Operation block, you must have a CONNECT-LAYER command in the Input-Layer block. Only layers that appear in CONNECT commands can appear in the CONNECT-LAYER command. Contact layers should not be listed. You can specify multiple layer names on one line or use multiple lines.

Arguments

layer1 A drawn or derived layer.
layer2 A drawn or derived layer.

Translation to Calibre

Connect-Layer = layer1 layer2 layer3 ... layerN

Translates to:

TEXT LAYER layer1 layer2 layer3 ... layerN
LABEL ORDER layerN ... layer3 layer2 layer1

Note: Note that the layer order is reversed in this translation, as required for the Label Order specification statement in Calibre.
IDTEXT

layer1 = IDTEXT layerNumber

Description

Specifies text layers to be used with SELECT BY LABEL. The IDTEXT command is used only with the SELECT BY LABEL \{[t] | [t']\} command. IDTEXT lets you add text to layers for identification purposes. These layers are independent of connectivity. You can select polygons by idtext names from a particular idtext layer.

Arguments

layer1 Layer name for text used by SELECT BY LABEL.
layerNumber GDS number of the layer used in the SELECT BY LABEL command.

Example

*INPUT-LAYER
metal1 = 1
IDTXT8 = IDTEXT 8
*END
*OPERATION
SELECT METAL1 BY IDTXT8 LABEL[T] VDD? MT1VDD
*END
TEXTSEQUENCE

TEXTSEQUENCE = layer1 layer2...

Description

The TEXTSEQUENCE command overrides the sequence given by CONNECT-LAYER. When multiple polygons on different nets overlap a label, then the net with the polygon whose layer appears last in the TEXTSEQUENCE list, if it is present, is labeled with the value of the net name. Text attaches first to the rightmost argument, then in order from right to left.

Arguments

layer

The name of the layer in the TEXTSEQUENCE command (excluding contact layers).

Calibre Format

*INPUT-LAYER
metal1 = 1
IDTXT8 = IDTEXT 8
*END
*OPERATION
SELECT METAL1 BY IDTXT8 LABEL[T] VDD? MT1VDD
*END

TEXTSEQUENCE = layer1 layer2 layer3 … layerN

Translates to:

LABEL ORDER layerN … layer3 layer2 layer1

Note that the layer order is reversed in this translation, as required for the Label Order specification statement in Calibre.

Operation Block Commands
AND

AND layer1 layer2 result-layer (OUTPUT c-name l-num {d-num})

Description

 Calculates the intersection of layer1 and layer2.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived polygon layer</td>
</tr>
<tr>
<td>layer2</td>
<td>A drawn or derived polygon layer</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>If the OUTPUT option is specified, the results of the operation are sent as</td>
</tr>
<tr>
<td></td>
<td>errors to a rule named c-name. The input layer names and l-num are written</td>
</tr>
<tr>
<td></td>
<td>as a rule comment. d-num is ignored.</td>
</tr>
</tbody>
</table>

Calibre Format

result-layer = layer1 AND layer2

Or

c-name { @ AND layer1 layer2 1-num
AND layer1 layer2
    }
ANDNOT

ANDNOT layer1 layer2 and-result not-result

Description

Produces the AND and NOT of two input layers in a single operation.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived polygon layer.</td>
</tr>
<tr>
<td>layer2</td>
<td>A drawn or derived polygon layer.</td>
</tr>
<tr>
<td>and-result</td>
<td>The result of layer1 AND layer2</td>
</tr>
<tr>
<td>not-result</td>
<td>The result of layer1 NOT layer2</td>
</tr>
</tbody>
</table>
AREA

\[
\text{AREA layer1 constraint result-layer \{OUTPUT c-name l-num \{d-num\}\}}
\]

**Description**

Produces layer1 polygons whose area conforms to the constraint.

**Arguments**

- **layer1**
  - A drawn or derived polygon layer
- **constraint**
  - **NE n1** — produces polygons whose area is not equal to n1.
  - **EQ n1** — whose area is equal to n1.
  - **RANGE n1 n2** — produces polygons with areas such that \(n1 < \text{AREA} < n2\). (Exclusive).
- **OUTPUT**
  - If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored

**Calibre Format**

\[
\text{result-layer = layer1 AREA layer1 >=n1 <= n2}
\]

Or

\[
c-name \{
\text{AREA layer1 >=n1<=n2}
\}
\]
CAT

CAT layer1 layer2 result-layer {OUTPUT c-name l-num {d-num}}

Description

Calculates the region formed by both layers minus the region shared by both layers. Performs the same operation as the OR command.

Arguments

layer1          A drawn or derived polygon layer
layer2          A drawn or derived polygon layer
OUTPUT          If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.
CONNECT

CONNECT layer1 layer2 BY connect-layer

Description

Defines a connection between overlapping objects on layer1 and layer2, where there is positive area overlap of layer1, layer2 and connect-layer. Connected objects are part of the same electrical net.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived layer.</td>
</tr>
<tr>
<td>layer2</td>
<td>A drawn or derived layer.</td>
</tr>
<tr>
<td>connect-layer</td>
<td>A drawn or derived layer. This specifies a contact layer.</td>
</tr>
</tbody>
</table>
CORNER

CORNER {{option}} layer1 relation-a {relation-b} {CORNER-SIZE n}
result-layer {OUTPUT c-name l-num {d-num}}

Description

Identifies polygon corners and creates boxes on corner vertices.

Arguments

option
A — Report 90° corners only.
B — Report 45° corners only.
C — Report any angle corners

layer1
A drawn or derived polygon layer.

relation-a
INSIDE — Creates boxes on the inside of polygons.
OUTSIDE — Creates boxes on the outside of polygons.

relation-b
INNER — Creates boxes on the concave side of corners
OUTER — Creates boxes on the convex side of corners
Used in combination with INSIDE or OUTSIDE, the INNER and OUTER options filter the output to produce only those boxes that meet both relation-a and relation-b.

CORNER-SIZE n
The size of the boxes created. Default value is 2 times the RESOLUTION, if this value is not specified.

OUTPUT
If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored

Calibre Format

There is no Calibre command corresponding to the Dracula CORNER command.
**COVERAGE**

**COVERAGE layer1 constraint windowSize stepSize result-layer**

**COVERAGE {NOT} layer1 constraint windowSize stepSize {OUTSQU square-layer} {SQSZ size-of-square} {result-layer} {OUTPUT c-name l-num}**

**COVERAGE {NOT} layer1 constraint RECT rectanglelayer {result-layer} {OUTPUT c-name l-num}**

**COVERAGE layer1 percentage windowSize stepSize result-layer**

**COVERAGE {NOT} layer1 LT/GT/RANGE percentage windowSize stepSize {OUTSQU square-file} {SQSZ size-of-square} {trapfile} {OUTPUT result-layer}**

**COVERAGE {NOT} layer1 LT/GT/RANGE percentage RECT rectanglelayer {trapfile} {OUTPUT result-layer}**

**Description**

Reports an error for rectangular window within which the ratio of the area of layer1 to the area of the window meets the constraint. Error flags are the rectangle in which the violation occurs. Results are merged prior to output.

**Arguments**

- **layer1**: A drawn or derived polygon layer.
- **constraint**: Specifies the ratio of the area of layer1 to the area of the specified boundary that must exist for the boundary polygon to be produced. The values p1 and p2 must be non-negative real numbers. The following constraints can be chosen:
  - **LT/LE p1** — Output windows in which the ratio of the area of layer1 to the area of the window is < or <= to p1.
  - **RANGE p1 p2** — Output windows in which the ratio of the area of layer1 to the area of the window is > p1 < p2.
  - **GT/GE p1** — Output windows in which the ratio of the area of layer1 to the area of the window is > or >= to p1.
  - **EQ p1** — Output windows in which the ratio of the area of layer1 to the area of the window is = to p1.
- **NOT**: Takes the complement of the constraint.
- **windowSize**: Specifies a window size within which the density check is computed.
- **stepSize**: Specifies the step size for moving the window.
- **OUTSQU square-layer**: Square-layer is a derived polygon layer containing squares of size size-of-square located at the center of each violating window.
SQSZ size-of-square  Specifies the size of squares on square-layer. If this parameter is not present, the default value is the stepSize.

RECT rectanglelayer  Specifies that, for each rectangle on rectanglelayer, check the ratio (area of layer1) / (area of rectangle), and output if the ratio meets the specified constraint.

OUTPUT  If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name.

Example

```
COVERAGE POLY LT 0.35 10 10 OUTSQU squareLayer SQSZ 2 PolyDen4 OUTPUT PolyDen4 64
COVERAGE POLY LT 0.35 10 10 OUTSQU squareLayer2 PolyDen5 OUTPUT PolyDen5 64
```

Calibre Format

```
rule-name (@ layer1 density constraint
    DENSITY layer1 constraint WINDOW w STEP s
)
```
DEVTAG

DEVTAG element[type] layer-b layer-c
  (for tagging from a defined ELEMENT BJT device)

DEVTAG[L] layer-a layer-b layer-c
  (for tagging from an intermediate layer)

DEVTAG[S] BJT[type] layer-d layer-e

DEVTAG[LS] layer-c layer-f layer-g

Description

Defines aliases for certain layers described in extract electrical properties by tagging multiple layers that are parts of a device with device numbers assigned by the device layer. You must define the device in the ELEMENT command before using this command.

The DEVTAG command links device numbers of the layers that form multiple emitter or collector bipolar devices in the layout database. It tags device number information onto the tagged layer (layer-b) from the tagging layer (device layer or layer-a) when polygons in the tagging layer overlap or touch polygons in the tagged layer. DEVTAG outputs tagged polygons to output layer-c.

Multiple tagging is allowed in bipolar devices where you can link a polygon to several devices. When the tagged layer (layer-b) is not directly overlapped to the device layer, an intermediate layer (such as a SIZE layer) can tag the device number information. You do not have to tag the device layer in the ELEMENT command.

While extracting parameters from a multiple-terminal device, make sure the function or use of all generated layers from which you are extracting parameters represents the exact device (for example, MOS: gate, source-drain, and body; BJT: collector, base, emitter, and so forth).

DEVTAG identifies the devices by the layer names you specify in the ELEMENT command. Thus, either the tagged layer you are extracting (layer-b) or the intermediate layers in the tagging sequence must correspond to the functional layers in the ELEMENT command.

Note: When you use more than one DEVTAG command in a rules file, they must be grouped together. Also, you cannot use any other command between a group of DEVTAG commands.

DEVTAG commands are used when extracting lateral PNP devices that have multi-collectors with one emitter area. Use this command to distribute the emitter area evenly to all the BJT transistors that share the same emitter in the SPICE output.

Arguments

element [type]  Type of bipolar device used in the ELEMENT BJT[type] command. This ELEMENT[type] is the tagging device for tagging directly from the bipolar device (ELEMENT BJT). For example, BJT[NV] for a vertical npn and BJT[LP] for a lateral pnp.
Examples

The following is an example of the type of lateral PNP devices. Since the collectors are used as device recognition layers, there are four BJT transistors defined.

```
PARSET BJT1 AREA PERI EA EP
ELEMENT BJT[LP] COLLPN COLLPN BASELPN EMITLPN
DEVTAG[S] BJT[LP] EMITLPN BMIT1
LEXTRACT BJT1 BMIT1 BY BJT[LP] BJTL	P
C1,C2,C3,C4: COLLPN
B: BASELPN
E: EMITLPN (Area is 100u)
```
Circuit Element Extract Commands

The circuit element extraction commands define electrical circuit elements. You use these commands to extract MOS devices, pads, and electrical parameters such as widths and lengths. When you define electrical nodes with CONNECT commands, circuit element extraction commands automatically generate a circuit netlist. You can use this extracted circuit to perform layout versus schematic checks (LVS).

Extract checks the devices defined in an ELEMENT command to verify that they are formed correctly. The first three characters you specify in an ELEMENT or PARASITIC command must be unique. Also, you cannot specify the same layer name more than once in the ELEMENT command.
EDGECHK

EDGECHK[O] layer1 \{ANGLE[90 | -90]\} LENGTH measurement value1 \{result-layer\}
\{OUTPUT c-name l-num \{d-num\}\}

Description

Checks the continuous path length of all edges, only Manhattan edges, or only Non-Manhattan edges on
the input layer. Flagged edges are extended inside the input polygon by one RESOLUTION by default,
or outside the polygon if the [O] option is specified. By default, all edges are included in the check. Use
the ANGLE option to specify only Manhattan or non-Manhattan edges.

Arguments

layer1 A drawn or derived polygon layer.
ANGLE[90] Only check Manhattan edges (parallel with X-Y axis).
ANGLE[-90] Only check non-Manhattan edges.
measurement Allowed measurements are LT, RA, GT.
[O] The output edge is extended inside the input polygon by default,
ooutside the input polygon if the [O] option is specified.

Calibre Format

Y = layer ANGLE angle_constraint
Z = PATH LENGTH Y measurement value1
result-layer = EXPAND EDGE Z INSIDE BY resolution_value
c-name \{
  COPY result-layer
\}

If ANGLE option is not present, translate as :

Z = LENGTH layer_constraint value1
result-layer = EXPAND EDGE Z INSIDE BY resolution_value c-name \{
  COPY result-layer
\}
If [O] option present, use OUTSIDE BY in EXPAND EDGE

For no ANGLE option:angle_constraint is ">=0 <=90".
For ANGLE[90]:angle_constraint is "< 0.1 > 89.9"
For ANGLE[-90]:angle_constraint is " > 0.1 < 89.9"
For [O]:In EXPAND EDGE, use OUTSIDE BY rather than INSIDE BY
ELEMENT BJT

\texttt{ELEMENT BJT\{\textit{type}\} layer-a layer-b layer-c layer-d \{layer-s\}}

\textbf{Description}

Defines bipolar devices.

ELEMENT BJT should not use terminal layer names having the three first letters “BJT.” In general, the device layer used for a vertical npn device must be the emitter region, because an npn can have multiple emitters. Also, the emitter region overlaps all three terminals of this device.

For the lateral pnp, the device layer must be the collector region because a pnp can have multiple collectors. The collector layer does not touch or overlap the emitter layer for its terminal connection. To overcome this, the emitter layer is sized to overlap the collector device layer and is stamped with node information.

\textbf{Note:} Once a layer has been defined as a device or terminal layer, it must not be changed by any operation (mentioned as an output layer) after the ELEMENT command.

\textbf{Arguments}

\begin{itemize}
  \item \textbf{type} Two-character code that denotes the type of BJT device. The first character must be any letter from A-Z. The second character is optional and can be any letter from A-Z or any number from 0-9 (except for 8). You can use this code to differentiate BJT devices. For example, vertical npn devices are type NV and lateral pnp devices are type PL.
  \item \textbf{layer-a} Device layer, one region per device. This layer defines the device region that touches or overlaps the bipolar conductor layers. For an npn device, this is usually defined by the emitter. For a lateral pnp device, this is usually defined by the collector.
  \item \textbf{layer-b} Terminal conductor layer for the collector terminal. This layer must carry node information.
  \item \textbf{layer-c} Terminal conductor layer for the base terminal. This layer must carry node information.
  \item \textbf{layer-d} Terminal conductor layer for the emitter terminal. This layer must carry node information.
  \item \textbf{layer-s} Substrate terminal layer. If you define this substrate layer, LVS checks for discrepancies or inconsistencies found in the layout or schematic. Discrepancies are reported in the LVS report file. You must also add a CDL definition for substrate.
\end{itemize}
Examples

In the following example, the `enpn` is the emitter region and device layer. Layers `covered`, `base`, and `np+` are the BJT terminal conductors and they must carry node information.

```
ELEMENT BJT[NV] enpn covered base np+
```

In the next example, `cpnp1` is the collector region and device layer.

```
ELEMENT BJT[PL] cpnp1 base buried oepnpl
```

This last example shows the `ELEMENT` command with a substrate layer defined. The name specified here is `CHANNEL`, but you can specify any name.

```
ELEMENT BJT[A] BJTW COLL BASE EMIT CHANNEL
```
ELEMENT CAP

ELEMENT CAP {[type]} layer-a layer-b layer-c {layer-s}

**Description**

Defines a capacitor device. ELEMENT CAP should not have terminal layer names with the three first letters “CAP.”

**Note:** Once a layer has been defined as a device or terminal layer, it must not be changed by any operation or mentioned as an output layer after the ELEMENT command.

**Arguments**

- **type**
  Two-character code that denotes the type of CAP device. The first character must be a letter from A-Z. A second character is optional and can be any letter from A-Z or a number from 0-9 (except 8). This code differentiates capacitor devices.

- **layer-a**
  Device layer, one region per device, that defines the capacitor region that touches or overlaps the capacitor conductor layers.

- **layer-b**
  Terminal conductor layer for one side of the capacitor. This layer must carry node information.

- **layer-c**
  Terminal conductor layer for the other side of the capacitor. This layer must carry node information.

- **layer-s**
  Substrate terminal layer. If you define this substrate layer, LVS checks for discrepancies or inconsistencies found in the layout or schematic. Discrepancies are reported in the LVS report file. You must also add a CDL definition for substrate.

**Example**

```
*OPERATION
.
.
ELEMENT CAP[PP] p12cap p12 p11
ELEMENT CAP[A] metpwel metal pwell
.
.
*END
```

**netlist:**

```
C1 sig1 sig2 3pf $ [PP] or $.MODEL=PP
C2 sig3 sig4 1 pf $ [A] or $.MODEL=A
```

LVS checks the capacitor type because the netlist has optional subtypes specified with brackets [] (or with the .MODEL command.)
ELEMENT DIO

\texttt{ELEMENT DIO \{[type]\} \\{layer-a\} \\{layer-b\} \\{layer-c\} \{layer-s\}}

\textbf{Description}

Defines the P-N junction diode. This command does not define an LPE element. \texttt{ELEMENT DIO} should not have terminal layer names with the three first letters “DIO”.

\textbf{Note:}

Once a layer has been defined as a device or terminal layer, it must not be changed by any operation or mentioned as an output layer after the \texttt{ELEMENT} command.

\textbf{Arguments}

- \texttt{type}
  
  Two-character code that denotes the type of DIO device. The first character must be a letter from A-Z. A second character is optional and can be any letter from A-Z or a number from 0-9 (except 8). This code differentiates diode devices. For example, P-diode to N-substrate devices are \texttt{[P]} type, and N-diode to Pwell devices are \texttt{[N]} type.

- \texttt{layer-a}
  
  Device layer, one region per device. This layer defines the diode region that touches or overlaps the anode and cathode conductor layers.

- \texttt{layer-b}
  
  Anode conductor layer for the anode terminal, usually the p+ diffusion or the P-well. This layer must carry node information.

- \texttt{layer-c}
  
  Cathode conductor layer for the cathode terminal, usually the n+ diffusion or the N-substrate. This layer must carry node information.

- \texttt{layer-s}
  
  Substrate terminal layer. If you define this substrate layer, LVS checks for discrepancies or inconsistencies found in the layout or schematic. It reports discrepancies in the LVS report file. You must also add a CDL definition for substrate.

\textbf{Example}

\begin{verbatim}
.
CONNECT-LAYER=nsub pwell nsd psd poly metal
*END
*OPERATION
.
AND psd diode pdio; identify the P+ pdio
CONNECT metal nsd BY cont
CONNECT metal psd BY cont
CONNECT nsd nsub BY ntnsub
.
ELEMENT DIO[P] pdio psd nsub
;For a p+ diode element to the n-substrate.
\end{verbatim}
ELEMENT IN?

ELEMENT device layer-a layer-b layer-c {layer-s}

This is the ELEMENT IN? syntax in the Operation Block of your rule file when the PARSET command syntax in the Description Block of your rule file is as follows:

PARSET = pname param1 param2 param3,...paramn

The ? in the ELEMENT IN? command stands for any letter of the alphabet, but the first 2 characters of the device name must start with “IN”.

Description

This command lets you use BOX devices to simulate inductors and extract some important parameters for the inductors, such as the number of turns or diameter of inner circle, and output them to the SPICE netlist file for supporting inductors in layout designs.

The first step is to use the PARSET command to specify what parameters will be extracted. There are four geometric primitives regarding inductors provided: ID, IN, IW, and IS; and one reserved parameter, LL. See the PARSET (page 794) command for more information on these four primitives and the reserved parameter.

Once a layer has been defined as a device or terminal layer, it must not be changed by any operation or mentioned as an output layer after the ELEMENT command.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>User-defined device name with three characters. The first two characters should be IN.</td>
</tr>
<tr>
<td>layer-a</td>
<td>Device layer, one region per device</td>
</tr>
<tr>
<td>layer-b</td>
<td>Terminal layer for one side of the inductor.</td>
</tr>
<tr>
<td>layer-c</td>
<td>Terminal layer for the other side of the inductor.</td>
</tr>
<tr>
<td>layer-s</td>
<td>Substrate terminal layer.</td>
</tr>
</tbody>
</table>

Example

Description Block:

PARSET = TEST ID IN IW IS LL
Operation Block:
ELEMENT IND INSYM ME4 ME5
LEXTRACT TEST INSYM BY IND INDPAR1 &
EQUATION LL=ID+IN+IW+IS
LPSESELECT[NY] IND output SPICE
ELEMENT LDD

```
ELEMENT LDD {{type}} layer-a layer-b layer-c layer-d {layer-e}
```

**Description**

Defines a lightly doped drain MOS element whose source and drain cannot be swapped. ELEMENT LDD should not have terminal layer names with the three first letters “LDD.”

A special LDD[X-] type stops the generation of pseudo gates when you run an LVS check. To disallow devices from forming pseudo gates, specify an ELEMENT LDD command where the first character type is an X and the second character type is either a letter from A-Z or a number from 0-9 (except 8). This allows LVS to locally stop the formation of pseudo gates in the layout and also allows LVS checking of mixed analog and digital circuits.

To perform LVS, you must define LDD devices on the schematic netlist. There is, however, no special element called LDD in SPICE or other standard simulation packages. Using Circuit Description Language (CDL), you can code LDD devices as MOS devices with the designator LDD[type] specified as a comment:

```
Mxxxxx nd ng ns {nb} mname {L=val1} {W=val2} $LDD[type]
```

**Note:** Once a layer has been defined as a device or terminal layer, it must not be changed by any operation or mentioned as an output layer after the ELEMENT command.

**Arguments**

- **type**
  
  Two-character code used to denote the type of LDD MOS device. The first character must be a letter from A-Z. The second is optional and can be any letter from A-Z or any number from 0-9 (except 8). This code differentiates MOS devices with different implants. For example, CMOS pull-up devices are P type and pull-down devices are N type. NMOS devices are E, D, and N types.

- **layer-a**
  
  Device layer, one region per device. This layer defines the device region that touches or overlaps the device conductor layers. For silicon-gate MOS, this is the channel region layer usually defined by the overlapped area of polysilicon and diffusion. For a gallium arsenic transistor, this is the channel region layer usually defined by the overlapped area of schottky metal and lightly doped diffusion. For metal-gate MOS, this is the gate region usually defined by the thin oxide mask.

- **layer-b**
  
  Gate conductor layer. For silicon-gate MOS, this is the polysilicon layer. For metal-gate MOS, this is the metal layer; for gallium arsenic, this is the schottky metal layer. This layer must carry node information.
layer-c
Lightly doped drain conductor layer. For silicon-gate MOS, this is the self-aligned lightly doped diffusion layer, usually defined by the lightly doped diffusion layer excluding the channel regions. For metal-gate MOS, this is the lightly doped diffusion layer. This layer must carry node information.

layer-d
Source conductor layer. For silicon-gate MOS, this is the self-aligned diffusion layer, usually defined by the diffusion layer Source conductor layer. For silicon-gate MOS, this is the self-aligned diffusion layer, usually defined by the diffusion layer.

layer-e
Substrate conductor layer. This substrate layer is for optional use in running LVS with checks for substrate connection mismatch. For MOS this is usually N-well, P-well, N-substrate, or P-substrate. If you specify this layer, it must carry node information.

Example

```
ELEMENT LDD[N] GNDX POLY1 NDDIF1 NDIF1 PWELL1
```
**ELEMENT MOS**

```
ELEMENT MOS {{[type]}} layer-a layer-b layer-c {layer-d}
```

**Description**

Defines metal or silicon-gate MOS devices. ELEMENT MOS should not have terminal layer names with the three first letters “MOS”.

A special MOS[X-] type stops the generation of pseudo gates when you run an LVS check. To disallow devices from forming pseudo gates, specify an ELEMENT MOS command where the first character type is an X and the second character type is either a letter from A-Z or a number from 0-9 (except 8). This allows LVS checking of mixed analog and digital circuits.

**Note:** Once a layer has been defined as a device or terminal layer, it must not be changed by any operation or mentioned as an output layer after the ELEMENT command.

**Arguments**

- **type**
  - Two-character code to denote the type of MOS device. The first character must be a letter from A-Z. The second is optional and can be any letter from A-Z or any number from 0-9 (except 8). This code differentiates MOS devices with different implants. For example, CMOS pull-up devices are [P] type and pull-down devices are [N] type. NMOS devices are [E], [D], and [N] types. See the note below on the special ELEMENT MOS[X-] type.

- **layer-a**
  - Device layer, one region per device. For silicon-gate MOS, this is the channel region layer usually defined by the overlapped area of polysilicon and diffusion. For a gallium arsenic transistor, this is the channel region layer usually defined by the overlapped area of schottky metal and lightly doped diffusion. For metal-gate MOS, this is the gate region usually defined by the thin oxide mask.

- **layer-b**
  - Gate conductor layer. For silicon-gate MOS, this is the polysilicon layer. For metal-gate MOS, this is the metal layer. For gallium arsenic, this is the schottky metal layer. This layer must carry node information.

- **layer-c**
  - Source/drain conductor layer. For silicon-gate MOS, this is the self-aligned diffusion layer, usually defined by the diffusion layer excluding the channel regions. For metal-gate MOS, this is the diffusion layer. This layer must carry node information.

- **layer-d**
  - Substrate conductor layer. This substrate layer is optional when you run LVS or LPE checking substrate connection mismatch. For MOS: usually N-well, P-well, N-substrate, or P-substrate. If you specify this layer, it must carry node information.
Example

In this example, the MOS[X] transistor forms pseudo gates for LVS. The MOS[XN] transistor is not allowed to form pseudo gates and remain as a transistor in an LVS run.

```plaintext
ELEMENT MOS[N] ngate poly nsd pwell
ELEMENT MOS[D] depchnl poly srcdrn
ELEMENT MOS[N] nchnl metal diff
ELEMENT MOS chnl poly srcdrn
ELEMENT MOS[P] oxide metal diff nsub
ELEMENT MOS[X] xgate poly nsd pwell
ELEMENT MOS[XN] xngate poly nsd pwell
```
ELEMENT RES

ELEMENT RES {{type}} layer-a layer-b {layer-s}

Description

Defines a resistor device (poly, diffusion, thin-film, etc.). This command does not define an LPE element. ELEMENT RES should not have terminal layer names with the three first letters “RES.”

Note:
Once a layer has been defined as a device or terminal layer, it must not be changed by any operation or mentioned as an output layer after the ELEMENT command.

Arguments

type
Two-character code that denotes the type of RES device. The first character must be a letter from A-Z. A second character is optional and can be any letter from A-Z or a number from 0-9 (except 8). This code differentiates diode devices. This code differentiates among resistor devices. For example, P-diffusion devices are [PD] type and poly devices are [PO] type.

layer-a
Device layer, one region per device. This layer defines the resistor region that touches or overlaps the resistor conductor layers.

layer-b
Terminal conductor layer usually defined by a conductor layer excluding the resistor region (layer-a). This layer must carry node information.

layer-s
Substrate terminal layer. If you define this substrate layer, LVS checks for discrepancies or inconsistencies found in the layout or schematic. LVS reports discrepancies in the LVS report file. (You must also add a CDL definition for substrate.)

Example 1

For a poly resistor element:

AND ipoly res pores; digitized resistor mask
NOT ipoly pores poly
ELEMENT RES[PO] pores poly

Example 2

For a p+ diffusion resistor element:

SELECT pdiff LABEL[R] R? pdifres
STAMP cont BY metal
ELEMENT RES[PD] pdifres cont

Example 3
LVS checks the resistor subtype because the CDL netlist specifies the option subtypes either with the brackets \[ \] or with the `.MODEL` command.

CDL netlist:

```cdl
R1 sig1 sig2 3k \[PO\] or $.MODEL=PO
R2 sig3 sig4 1k \[PD\] or $.MODEL=PD
```
ENC

ENC {{option1}} layer1 {&layer-a1}[O] layer2 {&layer-a2} measurement
{result-layer} {OUTPUT c-name l-num (d-num)} {&}

Description

Measures the distance between the outside of layer1 and the inside of layer2 boundaries and outputs the edge pairs that meet the constraints.

Arguments

layer1 A drawn or derived polygon layer.
layer2 A drawn or derived polygon layer.
opt1

The following options are common to WIDTH, ENC, EXT, and INT:

C — Only flags the edge-pair when the edges are parallel.
C’ — Only flags the edge pair when the edges are nonparallel.
P (-) — Flags segments of edges that project onto each other. Dracula definition is defined as: Two edges project if perpendicular lines from a referenced edge intersect the other edge. The referenced edge is the edge most closely aligned to the x or y axis.
P'(-) — Flags edges that do not project onto each other.
R — Constructs polygons from the projection of error edges. You must specify only the result-layer, and not specify OUTPUT when using this option. This option turns on the P option.
R' — Outputs error flags in polygon format so you can reuse error data. You can process the created layer with logical operations. The flags are one RESOLUTION unit wide (as specified in the Description block). You cannot specify an OUTPUT error cell.
S (+) — Flags violations on the polygon if the edges of the polygon are within a “square boundary” inside of the other polygon edge.
N — Only flag violations on polygons that are on different nets. Use the CONNECT command to connect nodes before using this option.
N' — Inverse of N option. Only flag violations on polygons that are on the same net.

(continued)
X — Checks the delta value in the x direction between two edges. The edges must project onto each other. Non-Manhattan data is not checked.

Y — Checks the delta value in the y direction between two edges. The edges must project onto each other. Non-Manhattan data is not checked.

The following options are for ENC command only:

E (+) — Flags polygons from layer1 that are totally outside polygons from layer2. Performs an enclosure test in parallel with other functions performed by the ENC command.

O (+) — Flags layer1 polygons and layer2 polygons that cut/overlap each other. Outputs edges from layer2 that are enclosed by layer1. Within a conjunctive rule, only layer2 (the enclosing layer) receives error flags.

T (+) — Flags outside segments of enclosed polygons (layer1) that touch inside segments of enclosing polygons (layer2).

V — Check all polygon edges, ignoring shielding by polygon containment of one layer inside another.

The O, G, and E options are only supported when directed as errors with an OUTPUT statement, they are not supported when directed to a layer. The O and G flags behave the same.
Unsupported Arguments

The following options are not supported:

U, U’, CORNER-CORNER n2, CORNER-EDGE n3

Calibre Format

The basic Dracula rule

ENC LAYER1 LAYER2 LT 2.0 OUTPUT ENC1 63

 Translates to Calibre format as:
ENC1 {
    ENC LAYER1 LAYER2 < 2.0 ABUT >0<90 SINGULAR
}

The mapping of Dracula options to Calibre format is shown in the table below. The following options
are common to WIDTH, ENC, EXT, INT

C   PARALLEL ONLY
C’  NOT PARALLEL
P   PROJECTING
P’  NOT PROJECTING
R   PROJECTING REGION
R’  In an ENC rule, R’ is translated using EXPAND EDGE layer OUTSIDE BY.

ENC[R’] layer1 layer2 LT d result

Translates to:

E1 = ENC [layer1] layer2 < d ABUT >0<90
E2 = ENC layer1 [layer2] < d ABUT >0<90
temp1 = EXPAND EDGE E1 OUTSIDE BY 0.01
temp2 = EXPAND EDGE E2 OUTSIDE BY 0.01
result = temp1 OR temp2
// Here 0.01 is the value of RESOLUTION

S   SQUARE
LT/LE Constraint is “< n1” or “<= n1”

ENC layer1 layer2 LT d

Translates to:

ENC layer1 layer2 < dABUT >0<90 SINGULAR

EQ N1 Constraint is “== n1”

RANGE n1 n2 Constraint is “> n1 < n2”, and does not include ABUT and SINGULAR parameters.

SELLT/SELLE ENC layer1 layer2 SELLT d result-layer

Translates to:

temp-edge-layer = ENC [layer1] layer2 < d
result-layer = layer1 WITH EDGE temp-edge-layer
SELGT/SELGE

ENC layer1 layer2 SELGT d result-layer

Translates to:

temp-edge-layer = ENC [layer1] layer2 <= d
result-layer = layer1 NOT WITH EDGE temp-edge-layer

SELNE

ENC layer1 layer2 SELNE d result-layer

Translates to:

temp-edge-layer = ENC [layer1] layer2 == d
result-layer = layer1 NOT WITH EDGE temp-edge-layer

N

NOT CONNECTED

N’

CONNECTED

X

For example, Dracula
ENC [X] layer1 LT 3 OUTPUT R2 64
INT [X] layer2 LT 3 OUTPUT R3 64

Translates to Calibre:

T0 = ANGLE layer1 == 90
T1 = ANGLE layer2 == 90

ENC T0 T1 < 3 PARALLEL ONLY PROJECTING ABUT > 0 < 90 SINGULAR
INT T0 T1 < 3 PARALLEL ONLY PROJECTING ABUT > 0 < 90 SINGULAR

Y

Similarly, Y option translates to ANGLE layerx == 0, with
PROJECTING and PARALLEL ONLY options enforced.
The X and Y options are mutually exclusive.

The following options apply to ENC only:

E

In the ENC rule, the E flag is translated by adding layer1 OUTSIDE
layer2 to the basic rule.

ENC[E] layer1 layer2 LT d OUTPUT result 63

Translates to:

result {
    ENC layer1 layer2 < d ABUT >0<90 SINGULAR
    layer1 OUTSIDE layer2
}
In the ENC rule, the O flag is translated by adding the INSIDE ALSO option.

ENC[O] layer1 layer2 LT d OUTPUT result 63

Translates to:

result {
ENC layer1 layer2 < d ABUT >0<90 SINGULAR INSIDE ALSO }

T  Modify ABUT >0< 90 to just ABUT < 90

V  MEASURE ALL
EQUATION

EQUATION K parameter = FORTRAN-expression {&}

Description

This property calculation command specifies the equations that compute device parameters from extracted layout geometric primitive parameters or other computed parameters.

Use this command in conjunction with the LEXTRACT (page 788) command. You must specify the parameter to compute and all EQUATION parameters in the PARSET (page 794) command in the description block.

If you conjunct equations, EQUATION evaluates them sequentially. All referenced parameters in equations must be computed in previous equations.

Do not use tab characters in the equation line.

Arguments

K

Specifies the fringe coefficient, and can apply to the same or different layer fringe extraction and overlap capacitance extraction with fringe consideration. You can use only DEPT and WIDT parameters as arguments. You must specify this equation first.

parameter

Parameter to compute.

FORTRAN-expression

The following expressions are accepted:

Numbers

Integer, real number, scientific notation

Variables

As defined in the PARSET command

Operations

Unary +, unary -, binary +, binary -, *, /, **

Parentheses

( )

Functions

LOG, EXP, MAX, MIN, SIN, COS, TAN, ASIN, ACOS, ATAN, SINH, COSH, TANH, SQRT, LOG10, INT, and ABS.

&

Conjunction of several equations.

Example 1

EQUATION CNUM=INT((W-(2*M+CW))/(CW+CS)+1)
where

- CNUM is the contact number
- W is the width of resistor body
- M is the margin of the resistor head and contact
- CW is the width of contact
- CS is the space of the contacts

**Example 2**

```
EQUATION ca=0.00005*area+1.0E-4*(peri-ovpr)+1.4E-4*ovpr+
     0.00005*area+1.0E-12*(2*peri-ovpr)+2.0

EQUATION ca=ca+4.5*(peri-ovpr)
```

**Example 3**

```
*.GLOBAL
*.BIPOLAR
*.RESVAL
*.SCALE
*.EQUATION
.PARAM
.SUBCKT test MINUS PLUS
Rrmfg1r0 PLUS MINUS
+(12.0+1.1*abs(2.34)) $[R1] $W=2u
.ENDS test
```
EXT

EXT{[option1]} layer1{&layer-a1}{[O]} {layer2{&layer-a2}} measurement
{result-layer} {OUTPUT c-name l-num {d-num}} {&}

Description

Measures the distance between the outside of layer1 boundaries, or the distance between the outside of layer1 and the outside of layer2 boundaries.

Arguments

layer1  A drawn or derived polygon layer.
layer2  A drawn or derived polygon layer.
option1

The following options are common to WIDTH, ENC, EXT, and INT:

C  — Only flags the edge-pair when the edges are parallel.

C'  — Only flags the edge pair when the edges are nonparallel.

P (-)  — Flags segments of edges that project onto each other. Dracula definition is defined as: Two edges project if perpendicular lines from a referenced edge intersect the other edge. The referenced edge is the edge most closely aligned to the x or y axis.

P'(-)  — Flags edges that do not project onto each other.

R  — Constructs polygons from the projection of error edges. You must specify only the result-layer, and not specify OUTPUT when using this option. This option turns on the P option.

R'  — Outputs error flags in polygon format so you can reuse error data. You can process the created layer with logical operations. The flags are one RESOLUTION unit wide (as specified in the Description block). You cannot specify an OUTPUT error cell.

S (+)  — Flags violations on the polygon if the edges of the polygon are within a “square boundary” inside of the other polygon edge.

N  — Only flag violations on polygons that are part of different nodes. Use the CONNECT command to connect nodes before using this option.

N'  — Inverse of N option. Only flag violations on polygons that are part of the same node.

(continued)
X — Checks the delta value in the x direction between two edges. The edges must project onto each other. Non-Manhattan data is not checked.

Y — Checks the delta value in the y direction between two edges. The edges must project onto each other. Non-Manhattan data is not checked.

The following options are for EXT command only:

E (+) — Flags a polygon that is totally enclosed by a polygon from the other layer.

O (+), G(+) — Flags layer1 and layer2 polygons that cut/overlap each other. Error flags cover segments within layer1 and layer2 that outline the overlapping area of the two polygons. The segments flagged are the segments of these two polygons that cut the edges of the polygons. Does not flag a polygon that fully encloses a polygon of the other layer.

T (+) — Flags outside segments of layer1 polygons that touch outside segments of layer2 polygons.

V — Check all polygon edges, ignoring shielding by polygon containment of one layer inside another.

H (+) — Flags the outside edges of notched layer1 polygons that fail the spacing check. A notch is a set of non-adjacent facing edges, or adjacent facing edges that create an external angle of less than 90 degrees. Use this option only with a single input layer.

The O, G, and E options are only supported when directed as errors with an OUTPUT statement, they are not supported when directed to a layer. The O and G flags behave the same.
Unsupported Arguments

The following options are not supported:

U, U', CORNER-CORNER n2, CORNER-EDGE n3

H option is supported, absence of H is not supported. The O, G, and E options are only supported when directed as errors with an OUTPUT statement, they are not supported when directed to a layer.

Calibre Format

The basic Dracula rule measurement

**LT/LE n1** — Flags an error if the two segments are spaced less than n1 for LT or less than or equal to n1 for LE. Does not flag an error if the segments touch.

**EQ n1** — Flags an error if the two segments are spaced equal to n1.

**RANGE n1 n2** — Flags an error if the two segments are spaced less than n2 and greater than n1. If you do not want acute-angle error flags, you must specify the RANGE measurement.

**SELLT/SELLE n1** — SELLT and SELLE output those layer1 polygons that would be flagged by using LT n1 or LE n1 with the same options. Specify a result-layer name and do not use the OUTPUT statement when specifying this option.

**SELEQ/NE n1** — SELEQ outputs those layer1 polygons that would be flagged by using EQ n1 with the same options. SELNE outputs those layer1 polygons that would not be flagged by using EQ n1 with the same options. Specify a result-layer name and do not use the OUTPUT statement when using this option.

**SELRA n1 n2** — SELRA outputs those layer1 polygons that would be flagged by using RANGE n1 n2 measurement with the same options. Specify a result-layer name and do not use the OUTPUT statement when specifying this option.

**SELGT/GE n1** — SELGT and SELGE outputs those layer1 polygons that would not be flagged when you specify a LE n1 or LT n1 measurement with the same options. Specify a result-layer name and do not use the OUTPUT statement when using this option.

**OUTPUT**

If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.

The [O] option after layer1 indicates to use the original layer, not the layer generated by the previous operation in the conjunctive rule. Use this option only with a conjunctive rule.

&

Creates a conjunctive rule. See “Conjunctive Rules” on page 724.
Ext Layer1 Layer2 Lt 2.0 Output Ext1 63

Translates to Calibre format as:

Ext1 {
    Ext Layer1 Layer2 < 2.0 ABUT >0<90 Singular
}

The mapping of Dracula options to Calibre format is shown in the table below. The following options are common to WIDTH, ENC, EXT, INT:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Parallel Only</td>
</tr>
<tr>
<td>C’</td>
<td>Not Parallel</td>
</tr>
<tr>
<td>P</td>
<td>Projecting</td>
</tr>
<tr>
<td>P’</td>
<td>Not Projecting</td>
</tr>
<tr>
<td>R</td>
<td>Projecting Region</td>
</tr>
<tr>
<td>R’</td>
<td>In a Ext rule, R’ is translated using Expand Edge layer Outside By.</td>
</tr>
</tbody>
</table>

Ext[R’] layer1 layer2 Lt d result

Translates to:

E1 = Ext [layer1] layer2 < d ABUT >0<90
E2 = Ext layer1 [layer2] < d ABUT >0<90
Temp1 = Expand Edge E1 Outside By 0.01
Temp2 = Expand Edge E2 Outside By 0.01
Result = temp1 OR temp2
// Here 0.01 is the value of Resolution

S | Square
LT/LE | < / <= n1
Ext layer1 layer2 Lt 2.0
Translates to:
Ext layer1 layer2 < 2.0 ABUT >0<90 Singular

EQ n1 | n1
RANGE n1n2 | n1 < n2, does not include ABUT SINGULAR parameters.
SELLT/SELLLE | Ext layer1 layer2 Sellt D result-layer
Translates to:
Temp-edge-layer = Ext [layer1] layer2 < d
Result-layer = layer1 WITH EDGE temp-edge-layer

N | Not Connected
N’ | Connected Space
The following options apply to EXT only:

E

In the EXT rule, the E flag is translated by adding \((\text{layer1 INSIDE layer2}) \text{ OR } (\text{layer2 INSIDE layer1})\) to the basic rule.

\[
\text{EXT}[E] \text{ layer1 layer2 LT } d \text{ OUTPUT result 63}
\]

Translates to:

\[
\text{result \{}
\text{EXT layer1 layer2 } < d \text{ ABUT } >0<90 \text{ SINGULAR (layer1 INSIDE layer2) OR (layer2 INSIDE layer1)}
\}
\]

O, G

In the EXT rule, the O and G flags are translated by adding \((\text{layer1 CUT layer2}) \text{ AND } (\text{layer2 CUT layer1})\) to the basic rule.

\[
\text{EXT}[E] \text{ layer1 layer2 LT } d \text{ OUTPUT result 63}
\]

Translates to:

\[
\text{result \{}
\text{EXT layer1 layer2 } < d \text{ ABUT } >0<90 \text{ SINGULAR (layer1 CUT layer2) AND (layer2 CUT layer1))}
\}
\]

OE

In the EXT rule, if both E and O options are present, the rule is most efficiently translated by adding the INSIDE ALSO option.

\[
\text{EXT}[EO] \text{ layer1 layer2 LT } d \text{ OUTPUT result 63}
\]

Translates to:

\[
\text{result \{}
\text{EXT layer1 layer2 } < d \text{ ABUT } >0<90 \text{ SINGULAR INSIDE ALSO } \}
\]

H

“H” behavior is default Calibre behavior for single layer rule. Use SPACE if “H” is not present

T

Modify ABUT >0<90 to just ABUT < 90

T

Remove ABUT constraint

V

MEASURE ALL
GROW

GROW layer1 dx dy dx1 dy1 result-layer {OUTPUT c-name l-num {d-num}}

Description

Oversizes right, left, top and bottom edges of a layer by individually specified amounts. This command is supported only in the case of dx=dy=dx2=dy2.

Arguments

layer1: A drawn or derived polygon layer

dx, dy, dx2, dy2: The amount to grow layer1 polygons. All values must be positive and equal. For all-angle polygons, Dracula grows all-angle edges by more than the specified grow amount. L-Edit grows all-angle edges by exactly the specified amount.

OUTPUT: If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored

Unsupported Arguments

This command is supported only in the case of dx=dy=dx2=dy2.

Calibre Format

result-layer = SIZE layer1 BY size_value
HOLE

HOLE layer1 x-extent y-extent result-layer

Description

Produces a polygon layer formed by the holes of the input layer.

Arguments

layer1 A drawn or derived polygon layer.
x-extent y-extent Holes must have the orthogonal extents larger than x-extent by y-extent to be included in the output.
result-layer Output layer that contains holes of layer1.

Example

Z = HOLES layer1
result-layer = RECTANGLE Z < x-extent BY < y-extent MEASURE EXTENTS
**INT**

\[
\text{INT}\{[\text{option1}]\} \text{layer1}\{&\text{layer-a1}\}{[0]} \text{layer2}\{&\text{layer-a2}\} \text{measurement} \\
\{\text{result-layer}\} \{\text{OUTPUT c-name l-num \{d-num\}}\} \{&\}
\]

**Description**

Measures the distance between the inside of layer1 boundaries and the inside of layer2 boundaries. Checks the amount by which polygons of two layers overlap.

**Arguments**

- **layer1**: A drawn or derived polygon layer
- **layer2**: A drawn or derived polygon layer
- **option1**: The following options are common to WIDTH, ENC, EXT, and INT:
  - **C**: Only flags the edge-pair when the edges are parallel.
  - **C'**: Only flags the edge pair when the edges are nonparallel.
  - **P (-)**: Flags segments of edges that project onto each other. Dracula definition is defined as: Two edges project if perpendicular lines from a referenced edge intersect the other edge. The referenced edge is the edge most closely aligned to the x or y axis.
  - **P'(-)**: Flags edges that do not project onto each other.
  - **R**: Constructs polygons from the projection of error edges. You must specify only the result-layer, and not specify OUTPUT when using this option. This option turns on the P option.
  - **R'**: Outputs error flags in polygon format so you can reuse error data. You can process the created layer with logical operations. The flags are one RESOLUTION unit wide (as specified in the Description block). You cannot specify an OUTPUT error cell.
  - **S (+)**: Flags violations on the polygon if the edges of the polygon are within a “square boundary” inside of the other polygon edge.
  - **N**: Only flag violations on polygons that are part of different nodes. Use the CONNECT command to connect nodes before using this option.
  - **N'**: Inverse of N option. Only flag violations on polygons that are part of the same node.

(continued)
option (continued)

X — Checks the delta value in the x direction between two edges. The edges must project onto each other. Non-Manhattan data is not checked.

Y — Checks the delta value in the y direction between two edges. The edges must project onto each other. Non-Manhattan data is not checked.

The following options are for INT command only:

T (+) — Flags inside segments of polygons of two layers that coincide.

V — Checks the inside edges of layer1 to the inside edges of all surrounding geometries of layer2. Check all polygon edges, ignoring shielding by polygon containment of one layer inside another.

measurement

LT/LE n1 — Flags an error if the two segments are spaced less than n1 for LT or less than or equal to n1 for LE. Does not flag an error if the segments touch.

EQ n1 — Flags an error if the two segments are spaced equal to n1.

RANGE n1 n2 — Flags an error if the two segments are spaced less than n2 and greater than n1. If you do not want acute-angle error flags, you must specify the RANGE measurement.

SELLT/SElle n1 — SELLT and SELLE output those layer1 polygons that would be flagged by using LT n1 or LE n1 with the same options. Specify a result-layer name and do not use the OUTPUT statement when specifying this option.

SELEQ/NE n1 — SELEQ outputs those layer1 polygons that would be flagged by using EQ n1 with the same options. SELNE outputs those layer1 polygons that would not be flagged by using EQ n1 with the same options. Specify a result-layer name and do not use the OUTPUT statement when using this option.

SELRA n1 n2 — SELRA outputs those layer1 polygons that would be flagged by using RANGE n1 n2 measurement with the same options. Specify a result-layer name and do not use the OUTPUT statement when specifying this option.

SELGT/GE n1 — SELGT and SELGE outputs those layer1 polygons that would not be flagged when you specify a LE n1 or LT n1 measurement with the same options. Specify a result-layer name and do not use the OUTPUT statement when using this option.

If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.

& Creates a conjunctive rule. See “Conjunctive Rules” on page 724.

The [O] option after layer1 indicates to use the original layer, not the layer generated by the previous operation in the conjunctive rule. Use this option only with a conjunctive rule.
Unsupported Arguments

The following options are not supported:

U, U', CORNER-CORNER n2, CORNER-EDGE n3

Calibre Format

The basic Dracula rule

INT LAYER1 LAYER2 LT 2.0 OUTPUT INT1 63

Translates to Calibre format as:

INT1 {
    INT LAYER1 LAYER2 < 2.0 ABUT >0<90 SINGULAR
}

C PARALLEL ONLY
C' NOT PARALLEL
P PROJECTING
P' NOT PROJECTING
R PROJECTING REGION
R' In an INT rule, R' is translated using EXPAND EDGE layer OUTSIDE BY.

INT[R'] layer1 layer2 LT d result

Translates to:

E1 = INT [layer1] layer2 < d ABUT >0<90
E2 = INT layer1 [layer2] < d ABUT >0<90
temp1 = EXPAND EDGE E1 OUTSIDE BY 0.01
temp2 = EXPAND EDGE E2 OUTSIDE BY 0.01
result = temp1 OR temp2
// Here 0.01 is the value of RESOLUTION

S SQUARE

LT/LE n1 < / <= n1

INT layer1 layer2 LT d

Translates to:

INT layer1 layer2 < d ABUT >0<90 SINGULAR

EQ n1 == n1

RANGE n1n2 n1 < n2, does not include ABUT SINGULAR parameters.
SELLT/SELE  INT layer1 layer2 SELLT d result-layer

Translates to:

temp-edge-layer = INT [layer1] layer2 < d
result-layer = layer1 WITH EDGE temp-edge-layer

N  NOT CONNECTED
N'  CONNECTED

The following options apply to INT only:

T  Modify ABUT >0< 90 to just ABUT < 90
V  MEASURE ALL
LENGTH

LENGTH layer1 {&layer1} measurement {OUTPUT c-name l-num {d-num}} {&}

Description

Produces all edges of layer1 whose length conforms to the constraint. The NOT option produces all layer1 edges not produced by the corresponding LENGTH operation.

Arguments

layer

A drawn or derived polygon layer

measurement

GT/GE/ LT/LE n1 — Output edges with length that are >, >=, <, <= the constraint value n1 respectively.

RANGE n1 n2 — Outputs edges with length that are greater than n1 and less than n2 ( > n1 < n2).

OUTPUT

If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored

&

Creates a conjunctive rule. See “Conjunctive Rules” on page 724.

Examples

EXT metal1 LT 0.3 &
LENGTH metal1 GT 10.0 OUTPUT M1.C 64

Calibre Format

LENGTH layer1 constraint
Chapter 21: HiPer Verify: Dracula Command Files

LEXTRACT

LEXTRACT pset layer-a {layer-b} BY ELEMENT[type]{pfname}{&}
LEXTRACT[ES] RES layer-c layer-d BY RES{[type]} &

Note:
L-Edit does not currently support extraction by node or parasitic extraction.

Description

Extracts geometric parameters from one or two layers of a BJT, CAP, DIO, MOS, or RES device. You specify the parameters to extract in a PARSET command in the description block. The number of input layers you specify with the LEXTRACT command depends on the primitive parameters you specify with the PARSET command. When you use the LEXTRACT command, you must also specify a PARSET command in the Description block.

The first layer you specify must be a device layer or a layer with the device number tagged on by a DEVTAG command. The second layer you specify can augment the parameters you are extracting from the first layer.

You should group all parameter extractions for each device type in a conjoined group. Specify only one LEXTRACT conjoined group with the EQUATION command for each device type. Similarly, specify only one conjoined group for each device type (element [type]).

You cannot place another command within a group of LEXTRACT commands. In addition, you need to group LEXTRACT, DEVTAG, LVSCHK, or LPECHK commands together. Do not use any other command within any group of these commands.

For bipolar devices, LEXTRACT extracts width and length, as well as the geometric parameters area and perimeter from one or more layers.

Antenna Checks

In certain applications, you might want to compare the ratio of device area to the layer area driving the devices. This check, called an antenna check, is useful when you want to ensure that induced capacitance, caused by a fabrication process or a circuit operation, does not cause the drive current or the transition time to become unstable.

Arguments

pset
Name of the parameter set PARSET to extract (specified in the description block). The name can have up to four alphanumeric characters, the first of which must be a letter.

layer-a
The first layer from which parameters are extracted. This layer must be either the device layer or a layer with a device number tagged by the DEVTAG command.

layer-b
The optional second layer from which parameters are extracted. It is used in relation to the first layer to extract geometric parameters such as overlapped areas and perimeters.
**Example**

Extracting width and length of a BJT:

```plaintext
*DESCRIPTION
...
MODEL=BJT[VN],VNPN BJT[LP],LPNP BJT[SP],SPNP DIO[OV],OVD
PARSET BJTS AREA OVPR L W
...
*END
*OPERATION
...
AND EMIT BURIED XYZ
NOT XYZ TRUBASE MNO
EXT[PR] VNPN CONT LT 6 DUMMY
ELEMENT BJT[VN] VNPN BURIED TRUBASE METAL1
ELEMENT BJT[LP] CLPNP CLPNP BURIED OLNPNP
ELEMENT BJT[SP] SUBPNP BULK EMIT TRUBASE
...
ELEMENT RES RESBODY TRUBASE
PARAMETER RES 1
...
LEXTRACT BJTS VNPN DUMMY by BJT[VN] VNPARA &
EQUATION W=OVPR &
EQUATION L=AREA/W
LPECHK BJTWPERCENT = 10.00 BJTLPERCENT = 11.0
LPECHK RESWPERCENT = 5.00 RESLPERCENT = 8.0
*END
```

To extract the length and width of a BJT, you must define a PARSET with a W and L. Then specify an LEXTRACT defining the equations and extraction layers. In this example, an EXT[PR] check is done between the device layer and a contact layer to produce an intermediate layer that can compute the periphery overlap (OVPR). The periphery overlap is treated as W while L is computed from AREA divided by W. If no existing layer can be used together with the device layer to produce the overlap periphery you want, you might need a new layer for that purpose.

In the SPICE file generated by the LPESELECT BJT command in conjunction with the LEXTRACT command, the bipolar SPICE file has the following format:

- **CA** = the area of the collector
- **BP** = the perimeter of the base
- **CP** = the perimeter of the collector
- **EA** = the area of the emitter
BA = the area of the base EP = the perimeter of the emitter
NOT

\[ \text{NOT layer1 layer2 result-layer (OUTPUT c-name l-num \{d-num\})} \]

**Description**

Calculates the region formed by layer1 minus layer2.

**Arguments**

- **layer1**: A drawn or derived polygon layer
- **layer2**: A drawn or derived polygon layer
- **OUTPUT**: If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.

**Calibre Format**

\[ \text{result-layer = layer1 NOT layer2} \]

Or

\[ \text{c-name \{ @ NOT layer1 layer2 l-num \}
\text{ NOT layer1 layer2 \}
\]
**OCTBIAS**

```
OCTBIAS layer1 BY n1 result-layer (OUTPUT c-name l-num)
```

**Description**

Cuts all Manhattan corners on the specified layer at a 45° angle, a distance of n1 from the corner.

**Arguments**

- `layer1`: A drawn or derived polygon layer.
- `n1`: The distance by which to cut Manhattan corners

**Examples**

```
OCTBIAS layer1 BY 1.0 oct_layer1 OUTPUT oct_layer1
```

**Calibre Format**

There is no Calibre command corresponding to the Dracula OCTBIAS command.
OR

OR layer1 layer2 result-layer {OUTPUT c-name l-num {d-num}}

**Description**

Calculates the region formed by the union of layer1 and layer2.

**Arguments**

- **layer1**: A drawn or derived polygon layer
- **layer2**: A drawn or derived polygon layer
- **OUTPUT**: If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.

**Calibre Format**

```
result-layer = layer1 OR layer2

Or

c-name { @ OR layer1 layer2 l-num
    XOR layer1 layer2
}"
```
PARSET

\[ \text{PARSET} = \text{pname}, \text{param1}, \text{param2}...\text{paramn} \]

**Description**

Specifies the names of a parameter set for a group of geometric primitives or computed parameters. To extract these geometric primitives and computed parameters, use the **LEXTRACT** (page 788) and **EQUATION** (page 774) commands in the operation block. If you specify LEXTRACT, you must also specify PARSET.

**Arguments**

- `pname` Specifies the parameter set name. Maximum length is four alphanumeric. The first characters must be a letter. Do not specify any of the following reserved names: MOSD, DIOD, DIO2, DIO3, CAPD, CAPF, BJTD, or RESD. Default parameter set name is CAPO.

- `param` Specifies the parameters in a set. The parameter limit is 18. PRE recognizes only the first ten parameter keywords and overwrites the last eight parameters with node and subnode information in order to save the data file size. The reserved parameter keywords used by SPICE must be limited to the first ten parameters.

**Geometric Primitives**

Primitives that relate to the layer(s) you specify in LEXTRACT. Extracted values relate to each device layer of layer-a in LEXTRACT. You cannot alter these primitives by using an equation.

- **ANG** Total internal angle
- **AREA** Area
- **PERI** Perimeter
- **W1** Overlapped perimeter to node 1 of second layer
- **W2** Overlapped perimeter to node 2 of second layer
- **OVAR** Overlapped area to second layer
- **OVPR** Overlapped perimeter to second layer
- **WIDT** Distance that the geometries on the layer run parallel
- **DEPT** Distance between the parallel geometries
- **TPR** Perimeter of touching geometries that cause fringe effect
- **CLL** Sum of fringe effect related to TPR
- **ID** Diameter of the inner circle. It’s possible that the diameter of X-direction is different from that of Y-direction. Whichever is smaller will be selected.
- **IN** Number of circles
- **IW** Width of each circle
- **IS** Spacing between circles

**Reserved Parameter Keywords**

Keywords reserved for parameter values that relate to specific devices. You usually specify these keywords for SPICE simulations. You change these keywords by using an EQUATION command.
Inductors

When you need to support inductors in layout designs, you must first use this PARSET command (in the description block commands section of your rule file) to specify what parameters will be extracted, followed by the ELEMENT IN? (see page 761) in the operation block of commands in your rule file.

There are four geometric primitives regarding inductors provided: ID, IN, IW, and IS; and one reserved parameter, LL. See the Geometric Primitives in “Arguments,” above for definitions of these four primitives.

**User-Defined Parameters**

Names other than reserved parameter keywords. The maximum length is four alphanumeric characters and the first character must be a letter.

**Inductors**

When you need to support inductors in layout designs, you must first use this PARSET command (in the description block commands section of your rule file) to specify what parameters will be extracted, followed by the ELEMENT IN? (see page 761) in the operation block of commands in your rule file.

There are four geometric primitives regarding inductors provided: ID, IN, IW, and IS; and one reserved parameter, LL. See the Geometric Primitives in “Arguments,” above for definitions of these four primitives.

**Example**

Extracting width and length of a BJT:

```
*DESCRIPTION
...
MODEL=BJT[VN],VPN BJT[LP],LPNP BJT[SP],SPNP DIO[OV],OVD
PARSET BJTS AREA OVPR L W
...
*END
```
**PLENGTH**

**PLENGTH layer1 measurement {result-layer} {OUTPUT c-name l-num { d-num}}**

**Description**

Produces layer1 edges where the length of a contiguous set of edges conforms to the constraint.

**Arguments**

- **layer1**: A drawn or derived polygon layer
- **measurement**: `NE/EQ n1` — Output edges with length that are not equal or equal to the constraint value n1 respectively.
- **RANGE n1 n2** — Outputs edge paths with length that are greater than n1 and less than n2 ( > n1 < n2).
- **OUTPUT**: If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored

**Examples**

```
INT [R] metal1 LT 0.5 thin_metal1
PLENGTH thin_metal1 RANGE 0 50.0 OUTPUT M1.D 64
```

**Calibre Format**

```
PATH LENGTH layer1 constraint
```
RECTCHK

**RECTCHK**

```
RECTCHK[option] layer {WIDLEN NE/EQ value1 value2} {result-layer} {OUTPUT c-name l-num}
```

```
RECTCHK[option] layer {WIDTH NE/EQ/LT/RA/GT value1 (value2)} {LENGTH NE/EQ/LT/RA/GT value1 (value2)} {result-layer} {OUTPUT c-name l-num}
```

**Description**

Selects rectangles with specified width and height measurements.

**Arguments**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R</strong></td>
<td>Select the shape only if it is a rectangle. By default, RECTCHK selects all shapes that are not rectangular in addition to the selected rectangles.</td>
</tr>
<tr>
<td><strong>A</strong></td>
<td>If both WIDTH and LENGTH are specified, by default the OR of the WIDTH and LENGTH constraints must be satisfied for a rectangle to be selected. If you use [A], then the AND of the WIDTH and LENGTH constraints must be satisfied for a rectangle to be selected.</td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>Check rectangles that form butted trapezoids. This option is not supported.</td>
</tr>
</tbody>
</table>

**WIDLEN**

Selects rectangles of dimension equal to or not equal to value1 by value2.

**WIDTH**

Selects rectangles with width meeting the specified measurement.

**LENGTH**

Selects rectangles with length meeting the specified measurement.

**OUTPUT**

If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment.
SCONNECT

SCONNECT upper_layer lower_layer BY contact_layer (LINK label)

Description

Establishes soft connections from the upper_layer polygons to lower_layer polygons through contact_layer polygons. Connections are unidirectional; node numbers are passed from upper_layer to lower_layer, but not in the other direction.

Connectivity information is passed from upper_layer to lower_layer, through contact_layer objects, where lower_layer objects have positive area overlap both contact-layer and upper_layer objects. Contact polygons receive node numbers from upper_layer geometries.

Arguments

- upper_layer: A drawn or derived polygon layer.
- Lower_layer: A drawn or derived polygon layer.
SELECT ANGLE

SELECT layer1 ANGLE [n1] output-layer {OUTPUT c-name l-num {d-num}}

Description

Select polygons based on the angles of the edges forming the polygon.

Arguments

layer1 A drawn or derived polygon layer.
ANGLE [n1] n1 can be one of the following:
   90 — Select polygons with all edges horizontal or vertical.
   45 — Select polygons with all edges horizontal or vertical or 45°, and containing at least one 45° edge.
   -90 — At least one edge is not horizontal or vertical.
   -45 — At least one edge is not horizontal, vertical or 45°.
OUTPUT If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.
SELECT BY LABEL

```
SELECT layer1 BY text_layer LABEL([t] | [t']) ( label| label-list) 
    result-layer {OUTPUT c-name l-num { d-num}}
```

**Description**

Produces all layer1 polygons that intersect the location of text objects on layer text_layer having the specified label name.

**Arguments**

- **layer1**: A drawn or derived polygon layer.
- **text_layer**: A layer containing text objects. This layer must have a corresponding IDTEXT statement in the input layer block.
- **t**: Selects objects that are labeled by the specified text.
- **t'**: Selects objects that are not labeled by the specified text.
- **label**: Any label, which can include meta characters such as ?.
- **label-list**: A list of up to a maximum of 20 labels. The list must be surrounded by braces and labels names should be space or comma separated. ({ label1 label2 ...}).
- **result layer**: Name of the output layer.

**Example**

```
IDTXT8    = IDTEXT     8
SELECT METAL1   BY    IDTXT8    LABEL[T]   VDD?    MT1VDD ;
```

**Calibre Format**

```
SELECT layer1 BY text layer LABEL([t] | [t']) ( label| label-list) 
    result-layer {OUTPUT c-name l-num { d-num}}
```

Translates to:

If [t] option
```
result-layer = layer1 WITH TEXT label text_layer 
    c-name { COPY result-layer }
```
If [t'] option
```
result-layer = layer1 NOT WITH TEXT label text_layer 
    c-name { COPY result-layer }
```

If label-list is specified, then create a temp result layer for each label in the label-list, then OR the temp layers together.
SELECT CONN

\[ \text{SELECT layer1 CONN layer2 result-layer \{OUTPUT c-name l-name \{d-num\}\}} \]

**Description**

Produces layer1 polygons that are connected to layer2. Select CONN first selects all layer1 polygons that are connected to layer2, then adds to the selection any other layer1 polygons that are on the same node as any polygons in the originally selected set. If either layer1 or layer2 do not have connectivity, then the result is an empty layer.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>A drawn or derived polygon layer.</td>
</tr>
<tr>
<td>layer2</td>
<td>A drawn or derived polygon layer.</td>
</tr>
</tbody>
</table>
SELECT CUT, TOUCH, ENCLOSE, OVERLAP

SELECT[N] {NOT} layer1 relation {[n1:n2]} layer2 result-layer {OUTPUT c-name l-num {d-num}}

Description

Selects polygons from layer1 that have a specified relation to layer2. Relations can be CUT, TOUCH, ENCLOSE, or OVERLAP. A range may also be specified.

Arguments

N
The N option turns on node-based selection. Select operations with the N option first select all polygons on layer1 that have the specified relation to layer2, irrespective of connectivity, then adds to the selection any other layer1 polygons that are on the same node as any polygon in the originally selected set.

layer1
A drawn or derived polygon layer

layer2
A drawn or derived polygon layer

relation
CUT — Produces layer1 polygons that have portions both inside and outside layer2. The NOT option produces those polygons not produced by the corresponding SELECT CUT operation.

TOUCH — Produces all layer1 polygons that are completely outside layer2 polygons but share an edge with layer2 polygons. The NOT option produces those polygons not produced by the corresponding SELECT TOUCH operation.

ENCLOSE — Produces layer1 polygons that completely enclose any layer2 polygon. The NOT option produces those polygons not produced by the corresponding SELECT ENCLOSE operation.

OVERLAP — Produces all layer1 polygons that have some or all area inside layer2 polygons or share an edge with layer2 polygons. The NOT option produces those polygons not produced by the corresponding OVERLAP operation.

[n1:n2]
Outputs layer1 polygons satisfy the relation with the specified range of layer2 polygons. The range values (n1 and n2) are inclusive and must be integers. Zero is not allowed.

OUTPUT
If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.

Calibre Format

result-layer = layer1 CUT layer2 >=n1 <= n2
result-layer = layer1 TOUCH layer2 >=n1 <= n2
result-layer = layer1 ENCLOSE layer2 >= n1 <= n2
result-layer = layer1 INTERACT layer2 >= n1 <= n2
SELECT INSIDE, OUTSIDE, HOLE

```
SELECT[N] {NOT} layer1 relation layer2 result-layer {OUTPUT c-name l-num {d-num})
```

**Description**

Selects polygons from layer1 that have a specified relation to layer2. Relations can be INSIDE, OUTSIDE or HOLE.

**Arguments**

- **N**
  - The N option turns on node-based selection. Select operations with the N option first select all polygons on layer1 that have the specified relation to layer2, irrespective of connectivity, then adds to the selection any other layer1 polygons that are on the same node as any polygon in the originally selected set.

- **layer1**
  - A drawn or derived polygon layer

- **layer2**
  - A drawn or derived polygon layer

- **relation**
  - **INSIDE** — Produces all layer1 polygons that are completely inside layer2 polygons. Touching from the inside is considered to be inside. The NOT option produces those polygons not produced by the corresponding SELECT INSIDE operation.
  
  **OUTSIDE** — Produces all layer1 polygons that are completely outside layer2 polygons. Touching from the outside is considered to be outside. The NOT option produces those polygons not produced by the corresponding SELECT OUTSIDE operation.

  **HOLE** — Produces all layer1 polygons that exactly fit inside holes of layer2 polygons. The NOT option produces those polygons not produced by the corresponding SELECT HOLE operation.

- **OUTPUT**
  - If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.

**Unsupported Arguments**

The N option for node based selection is not supported.

**Calibre Format**

```
result-layer = layer1 INSIDE layer2
result-layer = layer1 OUTSIDE layer2
```

There is no Calibre command corresponding to the Dracula HOLE command.
SELECT LABEL

SELECT layer1 LABEL net_name result-layer (OUTPUT c-name l-num { d-num})

Description

Produces all layer1 polygons that belong to the net having the specified net name. The connectivity on layer1 must be established through a connectivity operation. Net names are given by placing text labels (ports) on the layout.

Arguments

layer1 A drawn or derived polygon layer.
net_name The text label of the node to check.

Calibre Format

SELECT layer1 LABEL net_name result-layer (OUTPUT c-name l-num { d-num})

Translates to:

result-layer = NET layer1 net_name
c-name {COPY result-layer}
SELECT VERTEX

SELECT layer1 VERTEX [n1: n2] result-layer {OUTPUT c-name {d-num}}

Description

Produces layer1 polygons that satisfy the specified vertex count range.

Arguments

layer1 A drawn or derived polygon layer
layer2 A drawn or derived polygon layer
[n1:n2] The range of vertex counts. The command produces layer1 polygons that contain between n1 and n2 vertices, inclusive of n1 and n2. The values of n1 and n2 must be nonzero integers
OUTPUT If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.

Calibre Format

result-layer = layer1 VERTEX layer2 >n1 <= n2
SHRINK

```
SHRINK layer1 dx dy dx1 dy1 result-layer {OUTPUT c-name l-num {d-num}}
```

**Description**

Undersizes right, left, top and bottom edges of a layer by individually specified amounts. This command is supported only in the case of dx=dy=dx2=dy2.

**Arguments**

- **layer**: A drawn or derived polygon layer
- **dx, dy, dx2, dy2**: The amount to shrink layer1 polygons. All values must be positive and equal. For all-angle polygons, Dracula shrinks all-angle edges by more than the specified grow amount. L-Edit shrinks all-angle edges by exactly the specified amount.
- **OUTPUT**: If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.

**Unsupported arguments**

This command is supported only in the case of dx=dy=dx2=dy2.

**Calibre Format**

```
result-layer = SIZE layer1 BY size_value
```
SIZE

SIZE layer1 BY size_value {result-layer} {OUTPUT c-name l-num {d-num}}

SIZE layer1 BY size_value STEP step_value {result-layer} {OUTPUT c-name l-num {d-num}}

SIZE layer1 WITHIN layer2 BY size_value [STEP step_value] {result-layer} {OUTPUT c-name l-num {d-num}}

SIZE layer1 DOWN-UP BY size_value {result-layer} {OUTPUT c-name l-num {d-num}}

Description

Resizes a layer up or down. A positive size_value performs a size up, or grow, and a negative size_value performs a size down, or shrink.

Arguments

layer1 A drawn or derived polygon layer
size_value The amount to grow or shrink the layer1 polygons.
WITHIN layer2 Constrains layer1 to travel inside layer2 when performing the SIZE operation. size_value must be positive when this option is used.
STEP step_value The step size of the sizing process. If size_value is not evenly divisible by step_value then the last step may be smaller than step_value in order to size by exactly size_value.
DOWN-UP Performs a size-down and size-up of the specified amount in a single operation. If size_value is negative the effect is a size-up followed by size-down.
OUTPUT If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored.

Unsupported Arguments

Options B,L,N,O,P,R,S,T,W are not supported.

Calibre Format

result-layer = SIZE layer1 BY size_value INSIDE OF layer2 STEP step_value

Or

result-layer = SIZE layer1 BY size_value UNDEROVER
SNAP

SNAP layer1 TO grid-value {result-layer}

Description

Snaps the coordinates of the geometry to a specified grid. Results are output to another layer, or can be placed back on the original layer.

Arguments

layer1 A drawn or derived polygon layer
grid-value Grid to which to snap the layer. The value must be the multiple of the RESOLUTION specified in the Description block.
result-layer IAn optional target layer for the snapped layer. If not specified, the snapped results overwrite layer1.
STAMP

STAMP layer1 BY layer2 {OUTPUT ([options])c-name l-num ( d-num)}

Description

Transfers net identification information from layer2 polygons to layer1 polygons where layer1 polygons are overlapped by layer2 polygons from a single net. If a layer1 polygon is overlapped by two or more layer2 polygons from different nets, or not overlapped at all, then the layer1 polygon is an undefined net. Warning messages report missing or conflicting connections.

Arguments

layer1 A drawn or derived polygon layer.
layer2 A drawn or derived polygon layer.
contact-layer A drawn or derived polygon layer.
contact-layer An optional secondary keyword set, where name indicates an electrical node, that specifies the node number for floating polygons. Floating polygons are polygons on any specified lower_layer that are not connected to any upper_layer polygons. Floating polygons receive the node number of the electrical node having the specified name in the top-level cell.
Chapter 21: HiPer Verify: Dracula Command Files

WIDTH

WIDTH {{option1}} layer1 {&layer1}{{O}} measurement ANGLE = angle-opt
{result-layer} {OUTPUT c-name l-num {d-num}} {&}

WIDTH layer1 RECT = n3 BY n4 {result-layer} {OUTPUT c-name l-num {d-num}}

Description

Measures the distance between the inside of layer1 boundaries, and outputs edge pairs that meet the constraint.

Arguments

layer1 A drawn or derived polygon layer.

option1 The following options are common to WIDTH, ENC, EXT, and INT:

C — Only flags the edge-pair when the edges are parallel.

C’ — Only flags the edge pair when the edges are nonparallel.

P (-) — Flags segments of edges that project onto each other. Dracula definition is defined as: Two edges project if perpendicular lines from a referenced edge intersect the other edge. The referenced edge is the edge most closely aligned to the x or y axis.

P’(-) — Flags edges that do not project onto each other.

R — Constructs polygons from the projection of error edges. You must specify only the result-layer, and not specify OUTPUT when using this option. This option turns on the P option.

R’ — Outputs error flags in polygon format so you can reuse error data. You can process the created layer with logical operations. The flags are one RESOLUTION unit wide (as specified in the Description block). You cannot specify an OUTPUT error cell.

S (+) — Flags violations on the polygon if the edges of the polygon are within a “square boundary” inside of the other polygon edge.

X — Checks the delta value in the x direction between two edges. The edges must project onto each other. Non-Manhattan data is not checked.

Y — Checks the delta value in the y direction between two edges. The edges must project onto each other. Non-Manhattan data is not checked.
The following option options are for WIDTH command only:

\[D\] — Disables acute-angle checking.
\[L\] — The L option when used with SELEQ is useful for exact size checks.

WIDTH [L] VIA1 SELNE 3.0 result-layer

will output VIA1 polygons that are not exactly 3.0x3.0 in dimension.

- **LT/LE** \(n_1\) — Flags an error if the two segments are spaced less than \(n_1\) for LT or less than or equal to \(n_1\) for LE. Does not flag an error if the segments touch.
- **EQ** \(n_1\) — Flags an error if the two segments are spaced equal to \(n_1\).
- **GT/GE** \(n_1\) — Flags an error if the two segments are spaced greater than \(n_1\) for GT or greater than or equal to \(n_1\) for GE. Does not flag an error if the segments touch.
- **RANGE** \(n_1, n_2\) — Flags an error if the two segments are spaced less than \(n_2\) and greater than \(n_1\). If you do not want acute-angle error flags, you must specify the RANGE measurement.

- **SELLT/SELLE** \(n_1\) — SELLT and SELLE output those layer1 polygons that would be flagged by using LT \(n_1\) or LE \(n_1\) with the same options. Specify a result-layer name and do not use the OUTPUT statement when specifying this option.
- **SELEQ/SELNE** \(n_1\) — SELEQ and SELNE output those layer1 polygons that would be flagged by using EQ \(n_1\) or an NE \(n_1\) with the same options. Specify a result-layer name and do not use the OUTPUT statement when using this option.
- **SELRA** \(n_1, n_2\) — SELRA outputs those layer1 polygons that would be flagged by using RANGE \(n_1, n_2\) measurement with the same options. Specify a result-layer name and do not use the OUTPUT statement when specifying this option.
- **SELGT/GE** \(n_1\) — SELGT and SELGE outputs those layer1 polygons that would not be flagged when you specify a LE \(n_1\) or LT \(n_1\) measurement with the same options. Specify a result-layer name and do not use the OUTPUT statement when using this option.

- **angle-opt** 45, non-45, 90, or non-90 — Specifies that only edge-pairs at the specified angle are to be checked. When you use this option, the C option is automatically invoked.
- **OUTPUT** If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored
- **&** Creates a conjunctive rule. See “Conjunctive Rules” on page 724.
The WIDTH rule with the RECT option flags rectangles that are exactly n3 by n4 in dimension.

Unsupported Arguments

The following options are not supported:

CORNER-CORNER n2, CORNER-EDGE n3

Calibre Format

The basic Dracula rule

\[
\text{WIDTH LAYER1 LT 2.0 OUTPUT WIDTH1 63}
\]

Translates to Calibre format as:

\[
\begin{align*}
\text{WIDTH1} & \{ \\
& \quad \text{INT LAYER1} < 2.0 \ ABUT >0<90 \ SINGULAR \\
\} 
\end{align*}
\]

The mapping of Dracula options to Calibre format is shown in the table below. The following options are common to WIDTH, ENC, EXT, INT:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>PARALLEL ONLY</td>
</tr>
<tr>
<td>C'</td>
<td>NOT PARALLEL</td>
</tr>
<tr>
<td>P</td>
<td>PROJECTING</td>
</tr>
<tr>
<td>P'</td>
<td>NOT PROJECTING</td>
</tr>
<tr>
<td>R</td>
<td>PROJECTING REGION</td>
</tr>
<tr>
<td>R'</td>
<td>In a width rule, R' is translated using EXPAND EDGE layer OUTSIDE BY.</td>
</tr>
</tbody>
</table>

\[
\text{WIDTH[R']} \text{ LAYER1 LT d result}
\]

Translates to:

\[
\begin{align*}
\text{temp} & = \text{INT LAYER1} < d \ ABUT >0<90 \\
\text{result} & = \text{EXPAND EDGE temp OUTSIDE BY 0.01} \\
& \text{Here 0.01 is the value of RESOLUTION}
\end{align*}
\]

S  SQUARE
The following options apply to WIDTH only:

**LT/LE**  \( n1 < / \leq n1 \)

WIDTH layer1 LT d

Translates to:

INT layer1 < d ABUT >0<90 SINGULAR

**EQ/NE**  \( n1 == / != n1 \)

**RANGE n1 n2**  \( > n1 < n2, \) does not include ABUT SINGULAR parameters.

**SELLT/SELLE**  WIDTH layer1 SELLT d result-layer

Translates to:

temp-edge-layer = INT [layer1] < d result-layer = layer1 WITH EDGE temp-edge-layer

**angle-opt**  There is no direct translation to Calibre for angle-opt.

The following options apply to WIDTH only:

**D**  Do not include ABUT >0<90.

**L**  The L option when used with SELEQ is useful for exact size checks.

WIDTH[L] layer1 SELEQ d rule-name

Translates to:

Rule-name {RECTANGLE layer1 == d BY ==d }

WIDTH[L] layer1 SELNE d rule-name

Translates to:

Rule-name { NOT RECTANGLE layer1 == d BY ==d }

The WIDTH rule with the RECT option translates as follows:

WIDTH layer1 RECT = n3 BY n4 \{trapfile\} \{OUTPUT c-name l-num \{d-num\}\}

This translates to Calibre as:

c-name \{ RECTANGLE layer1 ==n3 BY ==n4\}
**XOR**

```
XOR layer1 layer2 result-layer (OUTPUT c-name l-num {d-num})
```

**Description**

Calculates the region formed by both layers minus the region shared by both layers.

**Arguments**

- **layer1**: A drawn or derived polygon layer
- **layer2**: A drawn or derived polygon layer
- **OUTPUT**: If the OUTPUT option is specified, the results of the operation are sent as errors to a rule named c-name. The input layer names and l-num are written as a rule comment. d-num is ignored

**Calibre Format**

```
result-layer = layer1 XOR layer2
```

Or

```
  c-name { @ XOR layer1 layer2 l-num
        XOR layer1 layer2
      }
```
Unsupported Dracula Commands

L-Edit does not support all of the Dracula commands, options or flags, particularly if they do not relate to DRC.

- **BREAK**
- **ABORT-MULT-STAMP**
- **ABORT-P-G-SHORT**
- **ABORT-SOFTCHK**
- **ACUTE-CLIP**
- **ALLOW-RECONNECT**
- **ARRAY**
- **ARRAY-ENABLED**
- **ATTACH**
- **ATTRIBUTE CAP**
- **ATTRIBUTE RES**
- **AUTOMATCH**
- **BASE-LAYER**
- **BLACKBOX-FILE**
- **BLOCK-NAME-ONLY**
- **BOUNDING-BOX**
- **BOXMAP**
- **BOX-M-FACTOR**
- **BOX-M-IGNORE**
- **CALCULATE**
- **CAP-POLARITY**
- **CARE-SPLIT-ORDER**
- **CELLBNDY**
- **CELLBOX-LAYER**
- **CELL-CHILD-TEXT**
- **CELL-ERROR-REP**
- **CELL-LIBRARY**
- **CHECK-MODE**
- **CHECK-PATH**
- **CIF-REF-FILE**
- **CMOS-NTYPE**
- **CMOS-PTYPE**
- **CNames-CHK-OFF**
- **CNames-CSEN**
- **CONN-ALL-WIRE**
- **CONN-INTER-WIRE**
- **CONN-OPEN-WIRE**
- **CONVERT-DATABASE**
- **CPOINT-FILE**
- **CSH-F-OPTION**
- CTEXT and ATTACH
- CUT
- CUT-TERM
- DATAFORMAT
- DD-RSH-COMMAND
- DEL-VIR-WIRE
- DENC
- DIODE-P-TO-G
- DIODESEQ
- DRACBATCH-EMAIL
- EBES-MODE
- EBES-STRIPE
- EBES-UNIT
- EBOX-SHORT-PIN
- ECONNECT
- EDTEXT
- ELCOUNT
- ELEMENT BOX
- ELEMENT PAD
- EMPTY-BLACKBOX
- EMPTY-ENC
- ENCBASEDOVLP
- ENCRECT
- END-MACRO
- ENVIRONMENT-MAX
- ERROR-PATH-WIDTH
- EXPAND-GATE-DELI
- EXPAND-INST-DELI
- EXPAND-NODE-DELI
- EXPLODE
- EXPORT
- EXTERM-NOWARN
- FASTSIZE
- FDELEMITER
- FILTER-LDD
- FILTER-REPORT
- FIX-INPUT-ORDER
- FLAT-GDS-SMALL
- FLATTEN
- FLATTEN-PWRGND
- FLG-EXPTH-OFFGRD
- FLOATCHK
- FNODE-CONNECT
- FRACTURE
- FRAME-EXCLUDE-LAYER
- FRAME-HOLE-LAYER
- FRINGE CAP
- GATE-RES-EXT
- GENRECT
- GEN-TEXT-FILE
- GEN-TEXT-FLTNODE
- GEN-TEXT-FRAME
- GEN-TEXT-LAYER
- GEN-TEXT-WIRE
- GEN-XRF-RPT
- GLOBAL-SCONNECT
- GPATHCHK
- GPATHDEF
- GROUND-NODE
- HCELL
- HCELL-COLUMN-1
- HCELL-FILE
- HCELL-IN-HCELL
- HCELL-MAX-PLACEMENTS
- HCELL-MAX-SEGMENTS
- HCELL-OPTIMIZE-PLACEMENTS
- HCELL-RULE
- HEDTEXT
- HIERARCHEN
- HIER-OUTDISK
- IGNORE-SCH-SUB
- IMPORT-LAYERS
- INAME-TOPDOWN
- INDISK
- INDISK-FILE
- INMAG
- INSTPIN-FILE
- INTERNAL-NODE
- INTERNODE-PREFIX
- KCELL-FILE
- KEEPDATA
- KEEP-INST-FPIN
- KEEP-SHORT-BJT
- KEEP-SHORT-CAP
- KEEP-SHORT-DIO
- KEEP-SHORT-MOS
- KEEP-SHORT-RES
- KEEP-TEXT
- KTEXT
- LAYER-FILE
- LAYER-FILE-TYPE
- LCONNECT
- LEGAL-CHAR-SET
- LIBRARY
- LIMIT-DRC-ERROR
- LIMIT-RUN-TO
- LINK
- LISTERROR
- LONG-SUM-LEN
- LOOP-TWO-CAP
- LOOP-TWO-RES
- LPECHK
- LPE-FORMAT
- LPE-GROUND
- LPE-PRECISION
- LPE-QUERY
- LPESELECT
- LUMPCAP
- LVL-PART
- LVSCHK
- LVS-OPTION
- LVSPLOT
- LVS-RPT-ONLY
- MAG-BEF-GRID
- MAGNIFY-IN
- MAGNIFY-OUT
- MAP-TEXT-FILE
- MATCH-P-G-BY-TYP
- MAX-FLASH
- MEBES-MODE
- MEBES-PROCESS
- MEBES-STRIPE
- MEBES-UNIT
- MEMORY-CORE
- MIN-FLASH
- MINI-SUMMARY
- MIRROR
- MODEL
- MOS-CAP
- MOS-M-COMPARE
- MULTICPU
- MULTI-GDS2-IN
- MULTILAB
- MULTI-PIN-FILE
- MULTIPLE-DISK
- MULTI-SHEETRES
- NDCOUNT
- NEIGHBOR
- NODE-FILE
- NODE-SELECT
- NOSIZE-CELL
- NO-SUB-TERM
- NOT-HCELL
- NRX-EXTRACT
- NRX-SAVE
- OAOUTTYPE
- OPTIMIZE-RULE
- OUT-ANNOTATED-LAYER
- OUTCELL
- OUTDISK
- OUTLIB
- OUTMAG
- OUTPUT-ONE-LAYER
- OVERLAP
- OVL
- OVPR-TOLERANCE
- PAD-LAYER
- PARALLEL-FILE
- PARAMETER CAP
- PARAMETER RES
- PARASITIC CAP
- PARASITIC DIO
- PARASITIC RES
- PAR-COMPARE
- PAROUTNAM
- PATHCHK
- PATH-W-OFFGRID
- PATH-W-ROUND
- PERI
- PG
- PGCONVERT
- PGE-COMPACT
- PGEFILE
- PGE-MERGE
- PGE-MERGE-STRIPE
- PGE-MIL
- PGE-OFFSET
- PGE-OVERLAP
- PINCAP-FILE
- PIN-TEXT-LAYER
- PLATE-SIZE
- PLOTLENGTH
- POINT-CONNECT
- POSTENC
- POWER-NODE
- PREFIX-PARASITIC
- PREINTERACTIVE
- PRIMARY
- PRINTFILE
- PROBE
- PROB-GEOM-LIST
- PROCESS-VAR
- PROGRAM-DIR
- PULL-DOWN
- PULL-UP
- RAM-CELL
- RAM-CORE
- RCONNECT
- RCONNECT-LAYER
- RCX-LAYERS
- RCX-MODE
- REDUCER
- RELOCATE
- REP-UNM-INST-PG
- RERUN-FILTER
- RLENGTH
- ROTATION
- ROUND-OFF-TOLER
- RSPFSELECT
- SAMELAB
- SCALEOUT and REV-SCALEOUT
- SCHEMATIC
- SELECT
- SELECT-MODE
- SIZE-CELL
- SIZE-MAX-VERTEX
- SIZE-MIN-WIDTH
- SIZENOT
- SKILL-LOADER
- SMART-LPE
- SMASH-CAP-TYP
- SMASH-FLOAT-RES
- SMASH-MOS-TYP
- SMASH-RES-TYP
- SMOOTH-SIZE
- SOFTCHK
- S-OPTION-CLASS
- SPF-MAP-FILE
- SPFSELECT
- START-MACRO
- STATUS-COMMAND
- SUBCKT-NAME
- SUBNODE-DELIM
- SUB-PROGRAM-NAME
- SUBSTRATE
- SUBTYPE-CSENS
- SUMMARY-ONLY
- SVS-LAYOUT
- SVS-SCHEMATIC
- SYSOUT
- SYSTEM or SYSIN
- TEMPORARY-LAYER
- TEXT-HEIGHT
- Tnames-CSEN
- TRANSISTOR
- TRIANGLE
- UNIT
- UNSPEC-LAY-PARA
- UNSPEC-para
- UNSPEC-SCH-PARA
- WINDEL
- WINDOW
- WINDOW-CUT
- WORK-DIR
- XBOX
- XCELL
- XDEVICE
- XVIA
- ZERO-SPAC-F-EQU
HiPer Verify provides a significant compatibility with the support of Assura™ DRC. SKILL commands that control the flow of the DRC checks are also supported so the command file can be run directly in HiPer Verify without modification. Layer names, purposes and parameters imported via L-Edit’s capability to import Virtuoso are used when you run Assura command files.*

This chapter provides a reference to compatible DRC command file formats in L-Edit and directions for setting up Assura rules and using the SKILL-language parser.

**Typographic Syntax**

The following typographic conventions are used in this chapter.

- **literal**: Nonitalic words indicate keywords that you must enter literally for rule, function, or option names.
- **field**: Words in italics indicate user-defined arguments for which you must substitute a name or a value.
- **|**: Vertical bars separate possible choices for a single argument. They take precedence over any other character.
- **[  ]**: Brackets denote optional arguments. When used with vertical bars, they enclose a list of choices. You can choose one argument from the list or choose none.
- **{  }**: Curly braces are used with vertical bars to enclose a list of choices. You must choose one argument from the list.
- **...**: An ellipsis (three dots in a row, ...) indicates that you optionally can repeat the previous argument.

**Terminology**

The basic command components described in this chapter.

- **rules**: Statements contained within rule sections in rule files and run-specific files are called rules.
- **parameters**: Statements that are preceded with the “?” character that are placed within a parameter section are called parameters.
- **functions**: Statements contained within rule files and run-specific files that are placed outside of rule or parameter sections are called functions.

* Assura is a registered trademark of Cadence Design Systems, Inc.
Rule Syntax and Formatting

Assura rules must conform to these requirements:

- Layer names, cell names and file names are case sensitive
- Layers must be defined before they are used
- Layers can be redefined any number of times

L-Edit will check for the following errors and resolve them as described:

- If an operation uses an undefined layer, L-Edit will warn that the layer name is not defined.
- If an edge layer is input for an operation that is only valid for polygon layer input, L-Edit will warn that there is a layer type mismatch.
- If an operation uses a layer that is not created because it’s the result of an unsupported operation, L-Edit will cancel the operation.
- If an operation or option is not supported, L-Edit will ignore the entire operation.

L-Edit provides these formatting benefits:

- Statements that appear in a rule can begin anywhere on a line and can span lines. In addition, statements and operations need not each begin on a new line.
- You can right-click on a layer name in the text editor to access the command “Go To Layer Definition” in the context-sensitive menu, which jumps the cursor to the line with the definition for that layer.
- #DEFINE, #IFDEF, #ELSE, and #ENDIF are supported as a mechanism of conditionally executing blocks of commands
- Commands may be commented out using C-Style (/* … */) and C++ Style (//) comments characters. C-Style comments may span multiple lines. C++ Style comments extend from the comment characters to the end of the line.

Structure of an Assura File

Assura DRC rules are written in logical order to test foundry design rules against the input layout data. The first step is identifying the physical design layers contained in the input layout data. The next step is to derive additional layers from the original input layers to allow the tool to test against specific foundry requirements. At that point a comprehensive set of commands to check geometric shapes and relationships in the layout to foundry process design rules can be applied. If needed, layout geometries can then be adjusted or resized to conform prior to other checking and the production of the final mask.

Layer Definition Rules

These rules identify the physical design layers contained in the input layout database. To allow the DRC rules to be used with different layout databases, you can have separate layerDefs subsections, for example, one listing L-Edit layer names and another listing GDSII layers.

Layer Derivation Rules

These logical operation rules are applied to the existing layers to derive new layers to test against specific foundry requirements. For example, you can derived MOSFET gate regions, well taps and substrate ties from the original layer information.
DRC Operations

DRC verifies that a layout conforms to the technology design rules. The Assura DRC checks every cell in the layout as if it the highest in the hierarchy so that errors are reported within a cell only once, rather than once per instance of the cell.

Command Usage

- Commands can be used as layer derivation statements or in rule check statements.
- Commands may be nested. For example:

```
outLayer = geomAnd( geomAbutting( L1 L2 ) L3 )
```

- A layer derivation statement writes the results of a command to a named layer. For example:

```
AND POLY ACTIVE GATE
```

- A rule check statement uses the OUTPUT keyword to direct the results of a command to the Verification Error Navigator. The name after the OUTPUT keyword is the rulename that will appear in the Verification Error Navigator. The number following the rulename is a layer number used by Assura. For example:

```
EXT GATE LT 2.0 OUTPUT GAS1 64
```

- Any command, not only dimensional check operations, may direct errors to the Verification Error Navigator. For example:

```
SEL VIA OUTSIDE METAL V1 OUTPUT V1 64
```
How To Run Assura Rules

To enable Assura DRC rules, invoke Tools > DRC Setup and add your Assura rule file(s) to the DRC Rule Sets to run list. (See “Setting Up DRC” on page 547 for complete dialog instructions.)

Assura Setup

The Setup Assura button accesses this multi-tabbed dialog.
Setup Assura — avParameters

Assura allows you to define certain *avParameters* that control the execution of the Assura run.

**dbuPerUU**

Specifies the *database units per user unit* and can be used to override the default value, which is taken from the **Setup Design > Technology** tab in L-Edit.

**inputLayout**

InputLayout controls which layerDefs block in the Assura rule file to use. Valid values are DF2, DFII, GD2, or GDII. Upper or lower case is acceptable.

The layerDefs section of an Assura file defines the original layers in the design. An Assura rule file may contain layer definitions based on layer names in the database using the “df2” option, or based on GDS layer numbers using the “gds2” option, as shown below.

```
layerDefs( "df2"
layer1
layer2
..)
layerDefs( "GDS2"
layer1
layer2
.. )
```
Setup Assura — Switches

Assura rule files can contain sets of operations to be included or excluded using a conditional statement that is executed based on the presence of a switch in the Switches tab of the Setup Assura dialog.

The `avSwitch()` function is used to test whether a user-defined switch has been defined and marked as present in the Switches tab. The `switch()` function is also supported as an alias for `avSwitch`.

For example, if the “doMetal1Rule” switch is defined in the Switches tab of the Assura Setup dialog and the Present column is checked, then the Metal1 drc rules in the conditional check will be run. The related Assura conditional statement would read as follows.

```plaintext
if( avSwitch("doMetal1Rules") then
    drc(Metal1 sep < 1 "Metal1 spacing < 1")
    drc(Metal1 width < 1 "Metal1 width < 1")
)
```
Setup Assura — userVariables

Assura rule files may refer to user parameters that are defined outside of the rule file, to control the flow of execution, or as input values input to DRC operations.

User variables are sent to the Skill parser verbatim, with quotes included, as:

```
name = value
```

For example, the variable TopMetal is defined with a value “M4.” The variable TopMetal could be used in a rule file as follows:

```
case(TopMetal
    ("M4"
        ... operations if TopMetal is "M4"
    )
    ("M5"
        ... operations if TopMetal is "M5"
    )
) ; case TopMetal
```
Setup Assura — techParams

Assura rule files may refer to technology parameters defined in the layout database, using the `techGetParam()` function. This function returns the value of a named parameter from the layout database. You define the technology parameters accessed by the `techGetParam()` function in the `techParams` tab of the Setup Assura dialog.

For example, technology parameters `libName`, `grid`, `NWL2`, `NWL1`, `ME1B`, and `ME1A` are defined in the `techParams` tab. The Assura rules file can access the value of these parameters as shown below:

```python
val = techGetParam(tfId "NWL1")
sprintf(txt "NWL1: NWell Minimum Width = %L" val)
drc( NWell width < val  "txt")
```

Setup Assura — Prolog Text

The Prolog Text tab allows you to enter Skill code to be executed prior to execution of the main Assura rule file. This can be used to setup the runtime environment for any items that do not fit within the previously described methods.

Initializing the Assura Setup

**Using Virtuoso Import for Assura Setup**

Importing a Virtuoso technology file that corresponds to the Assura file also imports the correct layer name and purpose settings and techParams for the design.
Note: You must do this technology import on an empty layout, before any layout is created. Then you can either create layout or import it via GDSII.

Parsing the File to Setup Assura

If you add the Assura file to the list of files in Setup DRC, and then perform a syntax check on the Assura file by opening the file and invoking Tools > Check Syntax (shortcut key F6), you can initialize the Switches, userVariables, and techParams tabs in the Setup Assura dialog, with all switches, user variables and technology parameters that are found in the Assura rule file.

Note, however, that switches are initialized as not present, and user variables and technology parameters are initialized without any values.
Functions By Category

Layer Definition Rules

Drawn layers are the layers you create in your layout. This section contains a description of the following layer definition section keywords and rules:

- **cellBoundary** (page 837)
- **layerDefs** (page 895)
- **layerName** (page 896)
- **pinText** (page 899)
- **textToPin** (page 901)
- **textToPin** (page 901)

Layer Derivation Rules

Derived layers are the layers the tool creates when it processes the original layers. Examples of derived layers are sized layers, combined layers you create with logical operations for gate-recognition purposes, or edge layers you create with `geomGetEdge` for device recognition purposes.

You can use a derived layer the same way you use an original layer. For example, in this simple rule:

```plaintext
gate = geomAnd( polysilicon diffusion )
```

creates a derived gate layer from the original `polysilicon` and `diffusion` layers. You can also use

```plaintext
x = gate
```

so the derived layer names `x` and `gate` refer to the same data created by the `geomAnd` rule.

The general format of the rules file used to define original layers and derive additional layers is as follows:

```plaintext
drcExtractRules(
  layerDefs(original layer definition rules)
  <layer derivation rules>
  <other rules related to the appropriate Assura verification tool>
)
```

Layer Derivation Rules Using Shape Selection

Shape selection rules let you derive layers by selecting shapes from one or two layers.

The selection criteria can be a logical operation performed upon, or a relationship between, shapes on one or two layers. It can also be based upon the geometries or characteristics of shapes, for instance angled edges or text strings, on one or two layers.

These shape selection rules are essential to derive layers from original layout layers in order to perform foundry-specified DRC checks and to create the device recognition and terminal layers necessary for LVS verification.

- **geomAnd** (page 852)
- geomAndNot (page 853)
- geomAvoiding (page 854)
- geomButting (page 856)
- geom ButtOnly (page 857)
- geomButtOrCoin (page 858)
- geomButtOrOver (page 859)
- geomCat (page 860)
- geomEnclose (page 866)
- geomContactCheck (page 864)
- geomEnclose (page 866)
- geomGetAngledEdge (page 867)
- geomGetEdge on page 869
- geomGetHoled on page 870
- geomGetLength on page 871
- geomGetNet on page 872
- geomGetNon45 on page 873
- geomGetNon90 on page 874
- geomGetNonRectangle on page 875
- geomStraddle on page 893
- geomGetTexted on page 878
- geomGetUnTexted on page 879
- geomGetVertex on page 880
- geomStraddle on page 893
- geomStraddle on page 893
- geomOutside on page 887
- geomOverlap on page 888
- Arguments on page 890
- geomSepOutside on page 890
- geomStraddle on page 893
- geomWidth on page 894
- geomgetpolygon

Layer Derivation Rules using Shape Sizing

- geomSize on page 891
- geomSize on page 891
- saveDerived on page 900
Layer Derivation Rules using Shape Generation

- `geomHoles` on page 882
- `geomHoles` on page 882
- `geomNoHoles` on page 885

Layout Display Rules

- `offGrid (DRC Offgrid Data Check)` on page 897

Layout Extraction Rules

- `geomConnect` on page 862
- `geomStamp` on page 892

Device Extraction Rules

- `extractCAP` on page 851

DRC Operations

DRC rules verify that a layout conforms to the minimum width, spacing, and other design constraints established for a given fabrication process or process variation.

- `area` (page 840)
- `coverage` (page 840)
- `sep` (page 842)
- `notch` (page 841)
- `sepNotch` (page 842)
- `width` (page 843)
- `enc` (page 841)
- `ovlp` (page 842)
- `venc` (page 843)

DRC Check Modifiers

DRC check modifiers apply restrictions on the edge relationships for the drc checks. You can use the check modifiers in any combination as long as they are not mutually exclusive (for example, `Parallel` and `notParallel`.)

```
[outLayer = ] drc( inLayer1 [inLayer2] check [modifiers] [message])
[errorLayer(outLayer)]
```

Note that if you use the `sameNet` or `diffNet` modifiers, you must first use `geomConnect` (page 862) to connect the input layers.
**Note:**

When you run a connectivity check in a check area window, the status of connected shapes outside the checking area is not included.

- **diffNet** (page 846)
- **sameNet** (page 848)
- **parallel** (page 848)
- **notParallel** (page 847)
- **withPerp** (page 849)
- **onlyPerp** (page 847)
- **opposite** (page 847)
- **app** (page 845)
- **length, lengtha, lengthb** (page 846)
- **lessShielded** (page 847)
- **shielded** (page 848)

**DRC Output Modifiers**

DRC output modifiers let you change the format of the output of a DRC check. Only one of these output modifiers below may be used in a DRC operation.

- **edge, edgea, edgeb** (page 846)
- **fig, figa, figb** (page 846)
Functions
cellBoundary

layerName = cellBoundary({root}["]")

Description

This keyword simply derives the minimum bounding box (MBB) of the top-level cell of the active design. It is used for area-based rules that rely on cell boundaries.

As of version 16, the file and cellname options are not supported, cellBoundary will always specify the rectangle bounding the top cell.
drc

[outLayer = ] drc( inLayer1 [inLayer2] check [modifiers] ) [errorLayer(outLayer)]

Description

Use the drc rule for design-rule checking operations with the available keywords and options.

If outLayer is specified, you must use errorLayer(outLayer) to save the error layer. If outLayer is not specified, an error layer is saved automatically.

Adding a Message to a DRC Check

You can include a message with any of the drc check keywords by entering text enclosed with double quotes as the last argument. L-Edit will display the text verbatim in the Error Navigator. In the following example,

angmet = geomGetAngledEdge( metal 0 < keep < 90 )
drc( angmet via enc < distance "via enclosure violation near angled metal" )
badvia1 = drc( metal via venc( width >20 encAll < 1.5 ) ( width <= 20 encAll <0.8 ) )
errorLayer( badvia1 "via enclosure violation near manhattan metal" )
badvia2 = geomAndNot( via metal )errorLayer( badvia2 "via straddling or outside metal" )

L-Edit will display via enclosure violation near angled metal or via enclosure violation near manhattan metal when it finds such an error.

Arguments

- **outLayer**
  - Writes the results of the DRC check to the layer you specify. The output modifiers define the result format.
  - If you do not specify an output modifier, drc writes the check results to outLayer in default form. In an area check, drc writes the original polygon to the output layer.

- **inLayer1**
  - Name of the first (or only) layer to process.

- **inLayer2**
  - Name of the second layer to process.
  - The order of the layers can be important, as several keywords and output modifiers give different results if you reverse the names of the first and second layers.

- **check**
  - Use the check keyword to specify the type of check and a dimensional limit, in one of these three syntax forms:
    - lower_limit op check_keyword op upper_limit
    - check_keyword op upper_limit
    - check_keyword op lower_limit
The lower and upper limits can be integers, floating point numbers, or expressions. You define both limits in the units used in the active file.

You must specify the upper limit for all checks except `area`. The lower limit is optional, with a default lower limit of less than or equal to 0.

You can use the following limit operators:
- `<`
- `<=`
- `>`
- `>=`
- `==`

### check keywords

_DRC Check Keywords_ (page 840) can be one of the following, which may also be used with modifiers:

- `area` (page 840)
- `coverage` (page 840)
- `sep` (page 842)
- `notch` (page 841)
- `sepNotch` (page 842)
- `width` (page 843)
- `enc` (page 841)
- `ovlp` (page 842)
- `venc` (page 843)

### modifiers

Use modifiers to restrict the scope of a check or to modify the output information. There are three types of modifiers:

- `check` (_DRC Check Modifiers_ on page 845)
- `output` (_edge, edgea, edgeb_ on page 846)
- `message` (_Adding a Message to a DRC Check_ on page 838)

### Examples

```
drc( active width < 2 sameNet notParallel)
drc( diffn sep < 3parallel diffNet edge)
```
**DRC Check Keywords**

The check keywords specify how the tool will evaluate geometric relationships in a layout.

**area**

Checks the area of individual shapes on a single layer and measures the entire shape. This check can have any combination of lower and upper limits. The area size is in square user units.

The only modifier area check uses is the `message` modifier.

**Example**

This example finds all shapes of otv whose area falls within the range greater than four but less than or equal to twelve).

```
drc( ovt 4< area <= 12 )
```

**coverage**

```
outLayer = drc(inLayer [boundLayer] coverage [boundary(x1 y1 x2 y2)]
[windowSize=wxy stepSize=sxy]
)
```

Measures the coverage density of a material against a specified coverage percentage, within every possible fixed-size window on the chip or on the given box on the boundary layer. You can alternately specify the `windowSize` and `stepSize` values to check the total coverage ratio.

You cannot use the two methods `boundLayer` and `boundary(x1 y1 x2 y2)` at the same time.

The `badSquares` option is ignored with a warning but the command is run. The `flat` and `heir` options are ignored silently.

**Arguments**

- **inLayer**: Layer for which coverage is calculated.
- **boundLayer**: Specifies the referenced check layer that makes the coverage rule check within the respective box of each individual shape in this boundary layer, instead of the box of the whole chip. `boundLayer` can specify more than one referenced check area.
- **coverage**: Keyword to specify drc coverage. The range check can have any combination of lower and upper limits. For example: `a < coverage < b`, `coverage < b`, `coverage > a`
- **boundary**: Use this keyword followed by two (x,y) coordinate positions that stand for the stand for the referenced check box to specify just one check area.

If you do not specify a `windowSize` and `stepSize`, L-Edit checks the total coverage value of the chip or boundary layer by default.

Merge checking will be performed if any portion of the moving window steps outside of the cell boundary. For efficient checking the `windowSize` value should be an integer multiple of `stepSize`. 
**windowSize**

Specifies the width and height of the scan window, in user units. Only a square window is supported.

**stepSize**

Specifies the number of windows within a scan window by setting the stepping distance of two successive positions of the window in either the X or Y direction.

For example, a cell starts at coordinates (0, 0) and the window is moving from left to right and bottom to top, the first window will cover a square whose diagonal goes from (0, 0) to (windowSize, windowSize) and the next window will cover (stepSize, 0) to (windowSize + stepSize, windowSize), etc.

The movement continues until the right edge of the window falls outside the cell boundary. Then the window moves back to the Y axis and starts at (0, stepSize) and so on.

*Note:* stepSize must divide evenly into windowSize.

**enc**

Measures the enclosure of shapes on the second layer by shapes on the first layer, where enclosure is measured as the distance between inside facing edges of shapes on the first layer to outside facing edges of shapes on the second layer.

enc verifies that first layer shapes enclose second layer shapes, so the order of the two required input layers is important. Each shape on a layer is checked against each shape on the other layer.

You must specify an upper or exact dimensional limit and you can optionally specify a lower dimensional limit, in user units.

There are cases where enc will fail to flag a violation.

```
In this case, you could use geomStraddle instead:

errlay1 = geomStraddle( A B )
```

```
In this case, you could use geomOutside instead:

errlay2 = geomOutside( A B )
```

**notch**

Checks a single layer only for the separation of edges that form notches, where a notch is the distance between outside facing edges of the same shape.
The notch check must have an upper dimensional bound and can also have a lower dimensional bound.

Example
This example finds all notches of metal2 whose dimensions are greater than 2.

\texttt{drc( metal2 notch > 2)}

\texttt{ovlp}

Measures the overlap of shapes on one layer by shapes on another layer, where overlap is the distance from an inside facing edge of a shape on one layer to an inside facing edge of a shape on the other layer. Note that because the keyword \texttt{ovlp} checks whether edges, not shapes, overlap, it can check shapes that are fully enclosed.

The order of the input layers is not important. Each shape on a layer is tested against each shape on the other layer.

You must specify an upper or exact dimensional limit and you can optionally specify a lower dimensional limit, in user units.

\texttt{sep}

Measures the separation between different shapes on the same layer or on two different layers, where separation is the distance between the outside facing edges of shapes.

If you define only one input layer, \texttt{sep} checks each shape on that layer against every other shape on the same layer. If you define two input layers, \texttt{sep} checks each shape on one layer against every shape on the other layer.

Note that \texttt{sep} does not check overlapping shapes; you must use \texttt{geomAnd} or \texttt{geomOverlap} for overlaps.

Example
This example finds gate-to-conn spacing violations when edges are between .4 and 3.2 apart.

\texttt{drc( gate conn .4< sep < 3.2 )}

\texttt{sepNotch}

This check performs both the \texttt{sep} and \texttt{notch} checks at once and is usually faster than running each check individually. It measures the separation between different shapes and also edges that form notches on individual shapes.

Shapes and edges must be on the same layer for this check, and only one distance can be checked for both separation and notches in a given rule.
Example

```
venc
[outLayer=] drc(encLayer viaLayer venc(viaEncList {enc1 <aa enc2 <bb enc3 <cc enc4 <dd}) [message])
```

Performs contact or via enclosure checks. As of version 16, L-Edit supports only the rectangular enclosure constraint option.

The `venc` test is performed on an enclosed polygon, usually a contact or a via, which must be rectangular with the edges parallel to the x or y axis. If an enclosed polygon is not a manhattan rectangle, it is ignored. An enclosed manhattan rectangle must be fully enclosed or all of its edges will be ignored.

Note that Assura DRC does not check enclosure with a single drc rule. Usually, a drc rule such as `drc(metal via enc < distance)` is accompanied by a `geomAndNot(via metal)` or a combination of `geomStraddle(via metal)` and `geomOutside(via metal)`. Similarly, the via enclosure check does not verify all via enclosure relationships, only those that are fully enclosed and orthogonal.

Arguments

```
outLayer The layer to which results of the DRC check are written.
encLayer Input layer that contains the enclosing shapes.
viaLayer Input layer with the rectangular via or contact shapes.
venc Keyword triggering the via enclosure check.
viaEncList List that contains measurements for the four possible widths.
width
```

```
[outLayer =] drc( inLayer width [modifiers] [withCornerTouch] )
```
Checks the width of individual shapes on a single layer, where width is the distance between inside facing edges of a given shape.

Arguments

**outLayer**
Output layer.

**InLayer**
Input layer.

**width**
Integer or floating-point number that specifies the width check. You must specify an upper dimensional limit, and you may optionally specify a lower dimensional limit.

**modifiers**
You can use any of the DRC check modifiers to limit or modify the output.

**withCornerTouch**
Use this keyword to find corner touch errors during a width check. *withCornerTouch* applies to the width function only, and can only be used with an upper limit as shown in the following syntax:

```
width op upper_limit
```

For example, this rule finds width with corner touch errors that are less than 5.

```
drc( met3 width < 5 withCornerTouch )
```
**DRC Check Modifiers**

**app**

Checks “apposition,” which is the projection of one edge onto the other. You specify the limits required for the apposition of edges to check.

As with *opp*, you must consider the application and meaning of apposition. When edges are parallel, the meaning is clear. When edges are not parallel to each other, the meaning of apposition depends upon the reference edge, which you cannot always predict. As a result, the apposition value is independent of the length of the edges to be checked. Apposition measures only within the range of the checking dimension.

You specify an apposition value as follows:

\[
\text{lower_limit op app op upper_limit}
\]

\[
\text{app op lower_limit}
\]

\[
\text{app op upper_limit}
\]

Operators can be any of the following: $<$, $<=$, $>$, $>=$, or $==$.

If angled edges diverge, apposition measures only up to the point where the separation equals the upper limit check dimension, as shown below.

![Diagram showing apposition calculations](image)

**Examples**

This example checks poly-to-diffn spacing of less than 2 but only flags error regions with an apposition greater than 3.

\[
drc( \text{poly diffn sep < 2 app > 3} )
\]

This example flags parallel metal edges that are less than 4 apart with an apposition less than 2.

\[
drc( \text{metal sep < 4 app < 2 parallel} )
\]

The next example flags edges of metal that are parallel to poly and less than 2 apart from poly, using an apposition that is greater than 5 but less than 20.

\[
drc( \text{metal poly sep < 2 5 < app < 20 parallel edgea} )
\]
diffNet

Modifies a check so it is performed only if the related edges are from polygons on different electrical nets. **diffNet** therefore only applies to two-layer checks or single-layer separations, and is effective only on connected layers.

For example, the following rule finds all shapes of **ovt** having an area that falls within the range greater than four but less than or equal to twelve.

```
drc( ovt 4< area <= 12 )
```

**edge, edgea, edgeb**

Normally a DRC check outputs the quadrilateral error region between the edges. These modifiers specify that the segments of the edges in an error become the output of the check, rather than the error region between the edges, as follows.

**edge**

Outputs the error segments of the edges of both layers in a two-layer check, or error segments of a single layer in a one-layer check.

*Note:* Edge results from a two-layer check are typically for display purposes only, not subsequent DRC operations.

**edgea**

In a two-layer check, the error segment of the edge of the first layer. In a single-layer check, the error segments of a single layer.

**edgeb**

The error segment of the edge of the second layer in a two-layer check. Does not apply to a single-layer check.

**fig, figa, figb**

Specifies that the output of the check is the polygons from which the edges originate rather than the edges themselves (or an error region between the edges.)

**fig**

Outputs the figures of both layers in a two-layer check, or the single layer in a one-layer check.

*Note:* **fig** results from a two-layer check are typically for display purposes only, not subsequent DRC operations.

**figa**

In a two-layer check, outputs polygons of the first layer. In a single-layer check, figures of the single layer.

**figb**

Outputs the polygon of the second layer in a two-layer check.

**length, lengtha, lengthb**

Specifies that a check is made only if the lengths of the individual edges in the check are within specified limits, using the following syntax.

```
lower_limit op length op upper_limit
length op lower_limit
length op upper_limit
```
Operators can be any of the following: $<, \leq, >, \geq$, or $\equiv$.

**length**

The lengths of edges from both layers in a two-layer check, or both shapes from the single layer in a single-layer check, must be within the modifier limits.

**lengtha**

The length of the edge from the first layer in a two-layer check must be within the modifier limits.

**lengthb**

The length of the edge from the second layer in a two-layer check must be within the modifier limits.

**lessShielded**

This modifier is used only with the `enc` keyword. It modifies an enclosure check to “look through” the wall when two shapes are apart from, or only butt, each other.

For example:

```
r334b = drc( endcap diff enc < 0.24 lessShielded )
```

Depending upon the use of the `shielded`, `lessShielded` or no modifier, `enc` check results can vary greatly.

- Using `enc` alone might or might not report these errors.
- Using `enc` with `shielded` will never report these errors.
- Using `enc` with `lessShielded` will always report these errors.

**notParallel**

Limits checking to edges that are not parallel to each other.

**onlyPerp**

Limits checking to edges that are perpendicular to each other.

**opposite**

Limits checking to only those edges that are opposite each other. For parallel edges, the portion of the checked edges is based on a perpendicular projection of each edge. For non-parallel edges, the meaning of “opposite” edges is not as clearly defined.

For nonparallel edges, the resulting error region depends on the perpendicular projections of both edges. The portion of an edge directly “facing” another edge can be different depending on the edge from
which you make the perpendicular projection, called the “reference edge.” In the example below, the blue lines show rectangle A as the reference edge and the black lines show B as the reference edge.

![Diagram showing reference edges](image)

The blue lines show A as the reference edge.
The black lines show B as the reference edge.

**parallel**

Limits checking to parallel edges.

**sameNet**

Modifies a check so it is performed only if the related edges are from polygons on the same electrical nets. sameNet therefore only applies to two-layer checks or single-layer separations, and is effective only on connected layers.

**shielded**

The shielded modifier changes DRC error reporting when other edges of the same layers intervene between the edges being checked—that is, when the error between edges is being “shielded.”

The effect of the shielded modifier varies depending on whether you have the opposite modifier specified also. Without the opposite modifier, the shielded modifier eliminates an error only if the intervening edge or shape completely shields the edges being checked. In the examples below, no error is formed for the left and middle figures, but the error for the right figure remains unchanged.

With the opposite modifier, an error is created between the edges being checked for each part of the edge pairs that have an unobstructed view of each other.
withPerp

Checks perpendicular edges in addition to the other important edge relationships.
errorLayer

```javascript
errorLayer( layer ["message"] )
```

**Description**

Defines the contents of a layer as errors and writes the layer name to the Verification Error Navigator.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer</td>
<td>User-specified error layer.</td>
</tr>
<tr>
<td>message</td>
<td>User-specified annotation.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
nAndPdiff = geomAnd( ndiff pdiff )
errorLayer( nAndPdiff "ndiff and pdiff must not overlap" )
```

This example uses `errorLayer` to specify the result of a `geomAnd` operation as an error.

See also `saveDerived` on page 900.
extractCAP

[outLayer =] extractCAP( "device" recLayer (pos "name") (minus "name")

Description

This function extracts capacitor devices from layout. Devices are not extracted if the number of physical terminals does not match the number of terminals specified.

As of version 16, the label, spiceModel, cellView and flagMalformed keywords are not supported.

Note: If you are going to create an extracted view, you must define the terminal names as they are defined in the symbol. In addition, the terminal layers pos, minus, and s must exist, and must be connected layers.

Arguments

<table>
<thead>
<tr>
<th>outlayer</th>
<th>Name of the device layer created by the extractCAP statement. If you do not specify a device output layer name, the tool creates a name as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>device_recLayer_CAP_number</td>
<td>where device and recLayer are the names you specify in the device and recLayer arguments, and number is an internally generated layer number.</td>
</tr>
<tr>
<td>device</td>
<td>Device name. Enter a character string enclosed in quotation marks. Spaces are not allowed.</td>
</tr>
<tr>
<td>reclayer</td>
<td>Recognition layer name. Each shape on this layer causes a separate device to be extracted.</td>
</tr>
<tr>
<td>pos, minus</td>
<td>Layer names of the capacitor terminals, where terminals are defined using the format shown below: name of the layer of the terminal, followed by an optional text string in parentheses that specifies the terminal name. The default terminal names are POS and MINUS.</td>
</tr>
<tr>
<td>s</td>
<td>Optional substrate connection, using the format shown below. The default terminal name is SUBSTRATE.</td>
</tr>
</tbody>
</table>

| layer [ ( "name" ) ] | |
|----------------------|
geomAnd

[outLayer =] geomAnd( inLayer1 [inLayer2] [edges] )

Description

Generates new shapes from the overlap of original shapes on the input layers.

You can specify one or two input layers to generate the areas common to both input layers. Note: If one of the input layers to geomAnd is an edge layer, the resulting output layer is an edge layer. You do not need to specify the edges option if one of your input layers is an edge layer.

Arguments

- **outLayer**: Output layer name.
- **inLayer1, inLayer2**: Input layer names. If you specify only one polygon layer, the input polygon layer must be an original graphics layer or the result of a geomSize operation with the unmergedOut option.
- **edges**: Specifies edge output. Note that geomAnd creates edges only when the edges of the shapes are colinear.

Example

gate = geomAnd( poly diff )
seloverlap = geomAnd( poly )

Before geomAnd

![Before geomAnd](image1)

After geomAnd

![After geomAnd](image2)
geomAndNot

```plaintext
[outLayer =] geomAndNot(inLayer1 inLayer2 [edges] )
```

**Description**

Generates new shapes from areas of the first layer that do not overlap any area of the second layer.

If one of the input layers to a geomAndNot is an edge layer, the resulting output layer is an edge layer. You do not need to specify the edges option if one of your input layers is an edge layer.

**Arguments**

- **outLayer**
  - Output layer name.
- **inLayer1, inLayer2**
  - Input layer names. Both must be specified.
- **edges**
  - Specifies edge output. GeomAndNot only creates edges when the edges of the shapes are colinear.
geomAvoiding

[outLayer =] geomAvoiding( inLayer1 inLayer2 )

Description

Selects shapes on the first layer that are completely outside, without abutting, shapes on the second layer. (See also “geomOverlap” on page 752.)

Arguments

| outLayer | Output layer name. |
| InLayer1, inLayer2 | Input layer names. Both must be specified. These can be either derived layer names or original graphics layer names. |
geomBkgnd

[outLayer =] geomBkgnd( )

Description

Generates a layer consisting of a single rectangle whose area encompasses all shapes on all layers of a cell. If a cell includes instances, the rectangle will not overlap the instances.

Arguments

outLayer

Output layer name of the merged background shape that encompasses all shapes on all layers of the entire design.
geomButting

```plaintext
[outLayer =] geomButting( inLayer1 inLayer2 )
```

**Description**

Selects polygons on the first layer that abut polygons on the second layer.

A polygon on the first layer can be coincident or overlapping with a single second-layer polygon to be evaluated as abutting.

Other relationships for the same shape do not affect the selection. For abut-only relationships, see the `geomButtOnly` (page 857) rule description.

The connection and limits options are not supported.

**Arguments**

- **outLayer**: Output layer name.
- **InLayer1, inLayer2**: Input layer names. Both must be specified.
**geomButtOnly**

```bash
[[outLayer =] geomButtOnly( inLayer1 inLayer2 [connection] [limits] )
```

**Description**

Selects polygons on the first layer that abut polygons on the second layer. This rule differs from `geomButting` (page 856) in that a shape on the first layer abuts only if none of its edges overlap polygons on the second layer.

The option `[connection]` is not supported.

**Arguments**

- **outLayer**: Output layer name.
- **InLayer1, InLayer2**: Input layer names. Both must be specified. These can be either derived layer names or original graphics layer names.
- **llimits**: Specifies the number, limit or range of abutting polygons with the keywords “keep” or “ignore” and the following operators:

  - `<`
  - `<=`
  - `>`
  - `>=`
  - `==`

  - **keep**—selects the shape on inLayer1 if the number of polygons abutted inLayer2 equals the limit or falls within the specified range.
  - **ignore**—rejects the shape on inLayer1 if the number of abutted inLayer2 polygons matches the limit or falls within the specified range.

**Examples**

```bash
badgate = geomButting( gate diffn 2 < ignore < 4 )
pulldown = geomButting( poly diffn sameNet )
```
**geomButtOrCoin**

\[
\text{geomButtOrCoin( inLayer1 inLayer2 [connection] [limits] )}
\]

**Description**

The geomButtOrCoin rule selects shapes on the first layer that abut or are coincident with shapes on the second layer. Butting and coincidence are defined as any edge-to-edge coincidence, regardless of whether the shapes overlap at the point of coincidence.

The options [connection] and [limits] are not supported.

**Arguments**

- **outLayer**
  - Output layer name.

- **inLayer1, inLayer2**
  - Input layer names. Both must be specified. These can be either derived layer names or original graphics layer names.
geomButtOrOver

[outLayer=] geomButtOrOver( inLayer1 inLayer2 [limits] )

Description

The geomButtOrOver rule selects shapes on the first layer that abut or overlap shapes on the second layer.

Butting is edge-to-edge coincidence where the shapes do not overlap at the point of coincidence. Overlap is defined as a common area between the shapes. The geomButtOrOver rule is a combination of the geomButting (page 856) and geomOverlap rules.

As of version 16, the option [connection] is not supported.

Arguments

- **outLayer**
  - Output layer name.

- **inLayer1, inLayer2**
  - Input layer names. Both must be specified.

- **limits**
  - Specifies the number, limit or range of abutting polygons with the keywords “keep” or “ignore” and the following operators:
    - `<`
    - `<=`
    - `>`
    - `>=`
    - `==`

  A shape on inLayer1 is not selected if there is not a shape on inLayer2 with the required relationship, even if the limits you specify encompass the value 0.

  If you specify a limit or range, at least one edge of the inLayer2 polygon must abut or overlap the inLayer1 polygon to select the inLayer1 polygon.

  - **keep**—selects the shape on inLayer1 if the number of polygons abutted or overlapping polygons on inLayer2 equals the limit or falls within the specified range.

  - **ignore**—rejects the shape on inLayer1 if the number of abutted or overlapping inLayer2 polygons matches the limit or falls within the specified range.

Example

```
mechanism = geomButtOrOver( big top sameNet 3 < ignore < 7 )
```
**geomCat**

```
[ outLayer =] geomCat( inLayer1 [ inLayer2 ...] )
```

**Description**

Combines shapes on the input layers without merging them.

If two or more shapes on the same layer are exactly the same and are in the same location, `geomCat` replaces them with a single shape.

**Arguments**

- **outLayer**
  The output layer, containing the unmerged shapes from all input layers.

- **InLayer1, InLayer2**
  Input layer names. You can specify one, two or more input layers. You cannot combine polygon layers with edge layers.
**geomCoincident**

```plaintext
[outLayer =] geomCoincident(inLayer1 inLayer2 [connection] [limits])
```

**Description**

Selects polygons on the first layer that have edges coincident with polygons on the second layer.

A shape on the first input layer is considered to be coincident with a shape on the second layer if any of their edges are coincident when the shapes overlap. For coincident-only relationships, see the rule `geomContactCheck` on page 864.

The options `connection` and `limits` are not supported.

**Prerequisites**

If you use the `connection` option with `geomCoincident`, you must first use `geomConnect` or `geomStamp` to transfer connectivity information to the input layers.

**Arguments**

- `outLayer` : Output layer name.
- `InLayer1, inLayer2` : Input layer names. You must specify two.
geomConnect

```
geomConnect( {via(viaLayer conLayer1 conLayer2 ...) | svia(viaLayer conLayer1 conLayer2 ...) | buttOrOver(conLayer1 conLayer2) | overlap(conLayer1 conLayer2)} ... [label(labelLayer conLayer) ...] [soft(conLayerSoft1 conLayerSoft2 ...) ...] )
```

**Description**

The *geomConnect* command defines layer connectivity by assigning all connecting shapes to nets for device extraction and subsequent netlist processing.

You can have multiple *geomConnect* statements in a rules file, but keep in mind that each new *geomConnect* statement redefines the entire connectivity used in the rules file from that point on. There are no incremental changes to connectivity.

As of version 16, the *probe* keyword is not supported.

**Note:**

*geomConnect* cannot process edge input layers—you must specify polygon input layers.

**Arguments**

**via**

Identifies a via layer and two or more conducting layers connected through the via layer. *geomConnect* connects shapes on the conducting layers that have a common area with shapes on the via layer.

**svia**

*svia* identifies a sequential via layer. A sequential via layer is a via layer and two conducting layers listed in their appropriate mask sequence.

You must specify the order of conducting layers so that they match the top-to-bottom sequence in the mask.

You can use the *viaLayer* parameter with both the via and svia keywords. The *viaLayer* keyword specifies the via layer name. If multiple via commands use the same *viaLayer*, they are processed as a single via clause.

**ButtOrOver**

 Specifies two connecting layers, conLayer1 and conLayer2, that are connected by abutment or overlap. Use this keyword for well-tie-to-wells and local interconnect.

You can use the *soft* keyword with connecting layers that are specified in *buttOrOver*. 
overlap

Similar to the geomOverlap function, specifies two connecting layers, \texttt{conLayer1} and \texttt{conLayer2}, that are to be connected by overlapping of the shapes. Use this keyword for well-tie-to-wells and local interconnect.

You can use the \texttt{soft} keyword with connecting layers that are specified in \texttt{overlap}.

label

Specifies a label layer and the conducting layers to which the label applies, and checks the text labels on the label layer against shapes on the conducting layers. If the origin of a label is inside a shape, \texttt{geomConnect} names the net for that shape.

\textbf{Note:} You must specify a \texttt{text} or \texttt{pinText} command for the \texttt{label} keyword to work.

If the origin of a label is positioned over more than one conducting layers, the label is applied from the first layer on the layer list.

You can place duplicate labels on a single net, and you can apply one label to a single net any number of times. However, if a single net has two or more names, \texttt{geomConnect} discards both names and flags them with a warning message.

If two or more nets have the same name, \texttt{geomConnect} adds a unique suffix to each of them.

If multiple labels with different names occur on the same net, \texttt{geomConnect} gives that net the name of the first label it finds.

If multiple nets have labels with the same name, \texttt{geomConnect} assigns the name to the first net it finds with that name and issues a warning message for each additional net with that name.
geomContactCheck

```plaintext
[ outLayer = ] geomContactCheck( inLayer [length_limits]
   [width_limits][ignore] )
```

**Description**

Selects all rectangular contacts on the input layer meeting the specified length and width criteria, and copies them to the output layer. For a shape to be interpreted as a rectangle, its edges must be orthogonal to the X and Y axes.

“length_limits” refers to the side along the major axis of the rectangle, while “width_limits” refers to the side along the minor axis.

If you include the `ignore` keyword, you can output all non-rectangular contacts, and also all rectangular contacts that fail to meet the specified length and width criteria. (To output only rectangles, see `geomStraddle` on page 893.)

**Arguments**

- **outLayer**: Output polygon layer name.
- **InLayer**: Input layer.
- **length_limits**: The keyword length used in a comparison to user units of length (the major axis of the polygon) using the standard comparison operators: `<`, `<=`, `>`, `>=`, `==`
  
  The value associated with `length_limits` must be equal to or greater than the value associated with `width_limits`.

  geomContactCheck defaults to the operator `&&` between `length_limits` and `width_limits`.

- **width_limits**: The keyword width used in comparison to user units of width (the minor axis of the polygon). The standard comparison operators are used (`<`, `>`, `==`, `>=`, and `<=`). The value associated with `width_limits` must be less than or equal to the value associated with `length_limits`.

  geomContactCheck defaults to the operator `&&` between `length_limits` and `width_limits`.
**ignore**

A keyword causing geomContactCheck to output all non-rectangular shapes. In the same operation, ignore outputs those rectangles that fail to meet the length and width criteria. Specifying geomContactCheck(inLayer ignore) is equal to geomGetPolygon(inLayer).

**ignore**—rejects the shape from inLayer1 if the number of inLayer2 polygons matches the limit or is within the specified range.

For example, 1 <= keep <= 3

**Example**

This example uses the keyword *ignore* to select all polygons; it also selects rectangular shapes that do not have length 6 and width 6.

badContacts = geomContactCheck( inLayer length==6 width==6 ignore )
**geomEnclose**

```c
[outLayer =] geomEnclose( inLayer1 inLayer2 [connection][limits] )
```

**Description**

Selects polygons on the first layer that entirely enclose polygons on the second layer. Other relationships for the same shape and coincident edges do not affect selection.

As of version 16, the option `[connection]` is not supported.

**Arguments**

- **outLayer**: Output layer name.
- **inLayer1, inLayer2**: Input layer names. You must specify two.
- **limits**: Specifies the number of enclosed polygons to keep or ignore using a limit or range value and the following operators:
  - `<`
  - `<=`
  - `>`
  - `>=`
  - `==`

  - **keep**—selects the shape from `inLayer1` if the number of enclosed `inLayer2` polygons matches the limit or is within specified the range.
  - **ignore**—rejects the shape from `inLayer1` if the number of `inLayer2` polygons matches the limit or is within the specified range.

  For example, `1 <= keep <= 3`

**Example**

```c
error = geomEnclose( nmos npn_fill 2 < ignore < 5 )
```
**geomGetAngledEdge**

```
[outLayer =] geomGetAngledEdge( inLayer [limits] [fig] )
```

**Description**

Selects edges or polygons from the input layer according to their angle relative to the X and Y axes.

**Arguments**

- **outLayer**: Output edge layer name, or polygon layer name if you use the `fig` option.
- **InLayer**: Edge or polygon input layer.
  - **Note**: If you use the `fig` option you must use a polygon layer as the input layer.
- **limits**: Specifies the angle an edge must have to be selected. Only a positive value between 0 and 90 is valid.

You can specify a limit or range with one of the following keywords:

- **keep**—Selects the polygons or edges if their angles relative to the X and Y axes match the limit or fall within the specified range.
- **ignore**—Rejects the polygons or edges if their angles relative to the X and Y axes match the limit or fall within the specified range.

**Note**: If you specify an inclusive range, the values must add up to 90.

You can use the following operators with a limit or range:

- `<`
- `<=`
- `>
- `>=`
- `==`

(continued)
The angle of an edge is measured independently of the polygon it is part of. The concepts ‘inside’ and ‘outside’ are not considered when measuring angles.

Although the angle or range of angles must be positive relative to the X axis, `geomGetAngledEdge` selects all edges with a positive or negative angle that conform to the specification relative to the X axis and Y axis, as shown in the following figure.

```
fig
```

You can use the `fig` keyword with polygon layer input to output the entire shape rather than just its selected edge.

**Examples**

```
keep == 35
ignore < 15 (also ignores > 70)
20 < keep < 40
0 <= ignore <= 90
manhattan = geomGetAngledEdge( gate keep == 90 fig )
app45 = geomGetAngledEdge( metal2 44 <= keep <= 46 fig )
```
geomGetEdge

[outLayer =] geomGetEdge( inLayer0 [op1 inLayer1] [op2 inLayer2])

Description

Selects edge segments from the edge or polygon input layer according to their relationships to polygons on other layers. You can specify a sequence of operators and layers to define a relationship between an edge of a shape on one input layer relative to the shapes on the other input layers.

You can use the data geomGetEdge extracts to check design rules but it is not intended for parameter and parasitic extraction.

As of version 16, the operators not over and over are not supported.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outLayer</td>
<td>Output layer name.</td>
</tr>
<tr>
<td>inLayer0</td>
<td>Edge or polygon input layer name. This layer contains the edge segments that geomGetEdge selects and copies to the output layer.</td>
</tr>
<tr>
<td>op1 inLayer1</td>
<td>A combination of operator and layer that defines when to select edges on inLayer0. inLayer1 must be a polygon layer. Possible values for are:</td>
</tr>
<tr>
<td></td>
<td>- butting—Edge coincidence between inLayer0 and inLayer1 where the layers do not overlap.</td>
</tr>
<tr>
<td></td>
<td>- coincident—Edge coincidence between inLayer0 and inLayer1 where the layers do overlap.</td>
</tr>
<tr>
<td></td>
<td>- outside—Edges of inLayer0 that are outside inLayer1 and are not butting.</td>
</tr>
<tr>
<td></td>
<td>- inside—Edges of inLayer0 that are inside inLayer1 and are not coincident.</td>
</tr>
</tbody>
</table>
geomGetHoled

\[ \text{outLayer=} \] geomGetHoled( inLayer )

**Description**

The geomGetHoled rule selects all polygons from the input layer that contain holes, where a hole (“donut”) is an area created when the perimeter of a polygon encloses an area that is not the polygon.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>outLayer</strong></td>
<td>Output layer name.</td>
</tr>
<tr>
<td><strong>inLayer</strong></td>
<td>Input layer name, either a derived layer name or a graphics layer name.</td>
</tr>
</tbody>
</table>
**geomGetLength**

```
[outLayer =] geomGetLength( inLayer limits [contiguous] [fig] [outLayer2 = unselectedEdges] )
```

**Description**

Selects edges from the input layer by their individual length.

*geomGetLength* generates individual edges as output regardless of the input format or *contiguous* option unless you use the *fig* option, which generates polygons.

As of version 16, the operator *contiguous* is not supported for a polygon layer.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>outLayer</strong></td>
<td>Output edge layer name, or polygon layer name if you use the <em>fig</em> option.</td>
</tr>
<tr>
<td><strong>InLayer</strong></td>
<td>Edge or polygon input layer.</td>
</tr>
<tr>
<td><strong>limits</strong></td>
<td>Use with the following keywords to specify a limit or range for the length of edges to be selected:</td>
</tr>
<tr>
<td></td>
<td>• <em>keep</em>—Selects the edge from <em>inLayer</em> if the length of a polygon edge matches the limit or falls within the specified range.</td>
</tr>
<tr>
<td></td>
<td>• <em>ignore</em>—Rejects the edge from <em>inLayer</em> if the length of a polygon edge matches the limit or falls within the specified range.</td>
</tr>
<tr>
<td></td>
<td>You can use the following operators with a limit or range:</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
</tr>
<tr>
<td></td>
<td>&gt;</td>
</tr>
<tr>
<td></td>
<td>&gt;=</td>
</tr>
<tr>
<td></td>
<td>==</td>
</tr>
<tr>
<td><strong>contiguous</strong></td>
<td>Specifies that the check is made on the sum of lengths of contiguous edges. Edges are contiguous if their ends meet at the same coordinate.</td>
</tr>
<tr>
<td><strong>fig</strong></td>
<td>You can use the <em>fig</em> keyword with polygon layer input or the contiguous option to generate polygon output rather than edge output.</td>
</tr>
<tr>
<td><strong>outLayer2</strong></td>
<td>Output edge layer name for unselected edges. All input edges that are not output on <em>outLayer1</em> are output on <em>outLayer2</em> if specified. This option is useful when multiple selections are being done on the same layer. It cannot be used with the <em>fig</em> option.</td>
</tr>
</tbody>
</table>
**geomGetNet**

```plaintext
[outLayer =] geomGetNet( inLayer "text1" ["text2"... ] )
```

**Description**

Selects polygons from the input layer by the name of the electrical net at the top-level cell to which they belong.

Any polygon that is part of a net having one of the specified net names is copied from the input layer to the output layer. The input layer must have been previously referenced in a geomConnect rule to extract the circuit connectivity. The net names you specify must refer to text labels.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>outLayer</code></td>
<td>Output layer name.</td>
</tr>
<tr>
<td><code>inLayer</code></td>
<td>Edge or polygon input layer name. This layer contains the edge segments that geomGetEdge selects and copies to the output layer.</td>
</tr>
<tr>
<td><code>text1, text2</code></td>
<td>The name of a net, in quotation marks, corresponding to a label in the circuit. You can use the wildcard operator * in the text string.</td>
</tr>
</tbody>
</table>
geomGetNon45

[outLayer =] geomGetNon45( inLayer [fig] )

Description

Selects edges from the input layer if they are not parallel to the X and Y axes of the circuit and are not at a 45-degree angles to the axes. Use the fig option to select polygons if any of the edges conform to these criteria.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outLayer</td>
<td>Output layer name. This can be a derived or a graphics layer.</td>
</tr>
<tr>
<td>InLayer</td>
<td>Edge or polygon input layer name. If you use the fig option, you must specify a polygon layer.</td>
</tr>
<tr>
<td>fig</td>
<td>Sends polygons to the output layer rather than edges.</td>
</tr>
</tbody>
</table>
geomGetNon90

[ outLayer = ] geomGetNon90(inLayer [fig])

Description

Maintains the net numbers of shapes in the output layer by selecting from the input layer those edges that are not parallel to the x and y axes of the circuit. If you specify the fig option, the rule selects complete shapes that meet this criteria.

Arguments

| outLayer | Optional output layer name. This can be a polygon or an edge layer. |
| InLayer  | Input layer name. This can be either a derived layer or a graphics layer, unless you specify the fig option, in which case it must be a polygon layer. |
| fig      | For polygon input layers, specifies that the figure containing the selected edge is passed to the output rather than to the edge itself. |
geomGetNonRectangle

[outLayer =] geomGetNonRectangle( inLayer [outLayer2 = unselectedShapes] )

Description

Selects all shapes that are not rectangles, where a non-rectangular shape has edges that are not parallel to the X and Y axes.

You can use the keyword unselectedShapes and a second output layer to output two layers in one rule.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outLayer</td>
<td>Output layer.</td>
</tr>
<tr>
<td>inLayer</td>
<td>Input layer.</td>
</tr>
<tr>
<td>outLayer2</td>
<td>Rectangle output layer name. This option must be followed by the keyword unselectedShapes.</td>
</tr>
</tbody>
</table>

Example

A and B are considered rectangles.

C is a non-rectangular polygon.
geomGetPolygon

[outLayer =] geomGetPolygon( inLayer [outLayer2 = unselectedShapes] )

Description

Note: This rule is identical to geomGetNonRectangle (page 875).

Selects all shapes that are not rectangles, where a non-rectangular shape has edges that are not parallel to the X and Y axes. You can use the keyword unselectedShapes and a second output layer to output two layers in one rule.

Arguments

- **outLayer**: Output layer.
- **inLayer**: Input layer.
- **outLayer2**: Rectangle output layer name. This option must be followed by the keyword `unselectedShapes`. 
geomGetRectangle

[ outLayer = ] geomGetRectangle( inLayer [ length_limits ] [ width_limits ] [ ignore ] ))

Description

Selects all rectangles on the input layer and copies them to the output layer, where a rectangle is defined as a shape having edges parallel to the X and Y axes.

You can set length and width criteria then select those rectangles that either satisfy or fail to satisfy the specified criteria.

length_limits refers to the side along the major axis of the rectangle, while width_limits refers to the side along the minor axis.

Arguments

outLayer Output layer.
inLayer Input layer.
length_limits The keyword length used in a comparison to user units of length (length being the major axis of the rectangle).

The standard comparison operators used are <, <=, >, >=, = and ==.

The value associated with length_limits must be equal to or greater than the value associated with width_limits.

The default operator between length_limits and width_limits is &&.

width_limits The keyword width used in a comparison to user units of width (the minor axis of the rectangle).

The standard comparison operators used are (<, <=, >, >=, = and ==).

The value associated with width_limits must be less than or equal to the value associated with length_limits.

The default operator between length_limits and width_limits is &&.

ignore Use this keyword to exclude rectangles that meet the specified length_limits and width_limits. All other rectangles are selected.

Note that geomGetRectangle(inLayer ignore) produces empty output, which will cause an error and terminate processing.
geomGetTexted

[outLayer =] geomGetTexted( inLayer textLayer [string ...] )

**Description**

Selects polygons from the input layer if the origin of the specified text string is over the area of the polygon.

**Arguments**

- **outLayer**
  
  Output layer.

- **inLayer**
  
  Input layer.

- **textLayer**
  
  Drawn layer with text strings.

  To select a polygon, you must specify one or more of the text strings on the textLayer, and the text string origin must be on a polygon of the input layer.

- **string**
  
  One or more text strings, each enclosed in quotation marks.

  If you omit the text string, this rule selects any polygon on the input layer with any text string. You can specify the wildcard character (*) in a string.

**Examples**

geomGetTexted(xx yy "pad") >> xx WITH TEXT yy "pad"

geomGetTexted(xx yy "pad*" >> xx WITH TEXT yy "pad?"

geomGetTexted(xx yy "pad1" "pad2" "pad3" ...) >> (xx WITH TEXT yy "pad1") OR (xx WITH TEXT yy "pad2") OR (xx WITH TEXT yy "pad3")
**geomGetUnTexted**

\[
\text{outLayer=} \text{geomGetUnTexted( inLayer textLayer string [string ... ] )}
\]

**Description**

Selects polygons based on the absence of a specified text string. Polygons are not selected, if the origin of the text string is over the any area of the polygon, including its edges or corners.

If you omit the text string, geomGetUnTexted selects any polygon on the input layer that has no text string.

**Arguments**

- **outLayer**: Output polygon layer.
- **inLayer**: Input polygon layer.
- **textLayer**: Drawn layer with text strings.
- **string**: One or more text strings on the `textLayer`, each enclosed in quotation marks. A polygon is selected when the text string origin is not on the polygon of the input layer.

Text strings can contain any characters allowed in a label. You can specify the wildcard character (*) in a string.

A text string can have multiple instances on different polygons. A single rule selects all polygons on the specified layer *except* those with the specified text string. You can use multiple `geomGetUnTexted` rules to deselect polygons on different layers with the same specified text string. You can also use multiple `geomGetUnTexted` rules to deselect a single polygon on an input layer many times, with different text strings.

**Examples**

**single string:**

\[ \text{geomGetUnTexted(xx yy ""pad""}) >> xx \text{ NOT WITH TEXT yy ""pad""} \]

**wildcard:**

\[ \text{geomGetUnTexted(xx yy ""pad*""}) >> xx \text{ NOT WITH TEXT yy ""pad?""} \]

**multiple strings:**

\[ \text{geomGetUnTexted(xx yy ""pad1"" ""pad2"" ""pad3"" ...}) >> \text{Temp = (xx WITH TEXT yy ""pad1""}) OR (xx WITH TEXT yy ""pad2""}) OR (xx WITH TEXT yy ""pad3""}) \]
\[ xx \text{ NOT Temp"} \]
geomGetVertex

[outLayer =] geomGetVertex( inLayer limits )

Description

Selects all shapes on the input layer with a vertex count matching the limit or within the range specified in the limits argument.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outLayer</td>
<td>Output layer.</td>
</tr>
<tr>
<td>inLayer</td>
<td>Input layer.</td>
</tr>
</tbody>
</table>
| limits     | Specifies the number of vertices a shape must have for geomGetVertex to select that shape. Specify the value by using a limit or range containing one of the following keywords:

- keep—Selects a shape if it adheres to the limit criteria.
- ignore—Rejects a shape if it adheres to the limit criteria.

You can use these standard comparison operators with a limit or range:

- `<`
- `<=`
- `>`
- `>=`
- `==`
**geomGrow**

```plaintext
outLayer = geomGrow( inLayer xValue yValue )
```

**Description**

Grows or shrinks shapes symmetrically in the horizontal and vertical directions.

A sizing value is applied equally in both directions to the X-axis and Y-axis, but the axes can be assigned two different sizing values.

Note also that you cannot apply a grow and a shrink parameter within one `geomGrow` rule.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>outLayer</strong></td>
<td>Output layer.</td>
</tr>
<tr>
<td><strong>inLayer</strong></td>
<td>Input layer. Must be a polygon layer.</td>
</tr>
<tr>
<td><strong>XValue</strong></td>
<td>The amount you want to size shapes in the horizontal direction. A positive number grows a shape, a negative number shrinks a shape, and zero performs no change. The maximum value is 10,000 and the minimum value is determined by the resolution.</td>
</tr>
<tr>
<td><strong>YValue</strong></td>
<td>The amount you want to size shapes in the vertical direction. A positive number grows a shape, a negative number shrinks a shape, and zero performs no change. The maximum value is 10,000 and the minimum value is determined by the resolution.</td>
</tr>
</tbody>
</table>
**geomHoles**

\[
\text{[outLayer =]} \text{ geomHoles( inLayer [area limits] )}
\]

**Description**

Generates shapes consisting of all the holes in polygons on the input layer.

As of version 16, *extent* limits are not supported.

Note that if *geomHoles* area is present in the rule deck, a *drc sep* of the same input layers must also be defined in the rule deck or an error will be generated. (*sep* (page 842) checks for minimum separation between input shapes and flags an error if the area of ring holes formed by two shapes is too small.)

```
geomHoles area calculations are illustrated below.
```

```
Note that geomHoles calculates area as an entire hole, regardless of any shapes the hole may contain.
```

```
In this example, the area of the green square donut hole is calculated as the full 4 * 4 =16, not 4*4 - 1*2=14.
```

```
So, if you are using an area limit, such that:
```

```
HolesLimit= geomCat( geomHoles( metal1 area <= 15 ) drc( metal1 sep >= 0.5) )
```

```
the green square donut hole will be selected.
```

**Arguments**

- **outLayer**
  - Output layer.

- **inLayer**
  - Input layer. Must be a polygon layer.
### area limits

Selects holes having an area that complies with a specified limit and writes them to the output layer.

The area limits operators are as follows, and area limit values must be positive:

- `<`
- `<=`
- `==`
geomInside

[outLayer =] geomInside( inLayer1 inLayer2 [connection] )

Description

Selects polygons on the first layer that are completely inside polygons on the second layer. Coincident polygons are considered inside in this check. If shapes on two layers are coincident on all sides, then this rule considers each layer inside the other.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outLayer</td>
<td>Output layer.</td>
</tr>
<tr>
<td>inLayer1, inLayer2</td>
<td>Input layer. As of version 16, must be a polygon layer.</td>
</tr>
</tbody>
</table>
**geomNoHoles**

```plaintext
[outLayer =] geomNoHoles( inLayer )
```

**Description**

Generates new polygons consisting of the original polygons of the input layer but without the holes.

(A hole is an area created when the perimeter of a polygon touches an area that is not part of the polygon, as shown below.

**Arguments**

- **outLayer** Output layer.
- **inLayer** Input layer.
**geomOr**

```plaintext
[outLayer =] geomOr( inLayer1 [inLayer2 ...] [edges] )
```

**Description**

Merges all shapes on one or more polygon input layers.

If one of the input layers is an edge layer, the resulting output layer is an edge layer, and you do not need to specify the edges option.

**Arguments**

- **outLayer**: Output layer.
- **inLayer1, inLayer2**: Input layers. As of version 16, must be a polygon layer.
- **edge**: Specifies edge output.

**Example**

This example merges all shapes on the active and thin_active layers:

```plaintext
poly = geomOr( active thin_active )
```
**geomOutside**

```bash
[outLayer =] geomOutside( inLayer1 inLayer2 )
```

**Description**

Selects polygons on the first layer that are completely outside polygons on the second layer. Abutting edges are considered outside.

As of version 16, the `connection` and `limits` options are not supported.

**Arguments**

- **outLayer**
  - Output layer.

- **inLayer1, inLayer2**
  - Input layers. Two must be specified and they must be polygon layers.
geomOverlap

[outLayer =] geomOverlap( inLayer1 inLayer2 {connection}[limits] )

Description

Selects polygons on the first layer that overlap polygons on the second layer.

Overlap occurs when a first-layer shape has any common area with a second-layer shape. Other relationships such as overlaps, abuts, or coincidence do not affect the selection.

As of version 16, the connection and limits options are not supported.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outLayer</td>
<td>Output layer.</td>
</tr>
<tr>
<td>inLayer1, inLayer2</td>
<td>Input layers. Two must be specified.</td>
</tr>
</tbody>
</table>
geomSepInside

[outLayer =] geomSepInside(deviceLayer maxDist contactLayer wellLayer
[anyOutside | allOutside] )

Description

Selects those shapes from deviceLayer that are separated by more than the maxDist value from all the shapes on contactLayer that are also contained within a shape on wellLayer. To be selected, a shape on contactLayer must also be at least partly contained within a shape on wellLayer.

Arguments

- **outLayer**: Output layer.
- **deviceLayer**: Input layer that is tested for shapes whose distance exceeds maxDist from the shapes on contactLayer.
- **maxDist**: The critical distance between any shape on deviceLayer and any shape on contactLayer.
- **contactLayer**: Input layer that is tested for shapes whose distance exceeds maxDist from shapes on deviceLayer.
- **wellLayer**: Input layer containing at least one shape within which a contactLayer shape is also found.
- **anyOutside**: Optional. Outputs the part of shapes that are outside the specified distance as well as shapes that are entirely outside that distance (comparable to geomAndNot.)
  - This is the default rule behavior.
- **allOutside**: Optional. Outputs only those shapes that are entirely outside the specified distance (comparable to geomOutside.)

Examples

This example selects those shapes on via that are more than 125 um from any shape on channel. A shape on channel is not selected unless it is contained within a shape on nwell:

res_loss_high = geomSepInside( via 125 channel nwell )
**geomSepOutside**

```
[outLayer =] geomSepOutside( deviceLayer maxDist contactLayer wellLayer
                                             [anyOutside | allOutside] )
```

**Description**

Selects those shapes from `deviceLayer` that are separated by more than the `maxDist` value from all the shapes on `contactLayer` that are also contained within a shape on `wellLayer`. To be selected, a shape on `contactLayer` must also be at least partly contained within a shape on `wellLayer`.

**Arguments**

- **outLayer**: Output layer.
- **deviceLayer**: Input layer that is tested for shapes whose distance exceeds `maxDist` from the shapes on `contactLayer`.
- **maxDist**: The critical distance between any shape on `deviceLayer` and any shape on `contactLayer`.
- **contactLayer**: Input layer that is tested for shapes whose distance exceeds `maxDist` from shapes on `deviceLayer`.
- **wellLayer**: Input layer containing at least one shape within which a `contactLayer` shape is also found.
- **anyOutside**: Optional. Outputs the part of shapes that are outside the specified distance as well as shapes that are entirely outside that distance (comparable to `geomAndNot`).
  - This is the default rule behavior.
- **allOutside**: Optional. Outputs only those shapes that are entirely outside the specified distance (comparable to `geomOutside`).

**Examples**

This example selects those shapes on `via` that are more than 125 um from any shape on `channel`. A shape on `channel` is not selected unless it is contained within a shape on `nwell`:

```
res_loss_high = geomSepInside( via 125 channel nwell )
```
geomSize

[outLayer =] geomSize(inLayer size [edges][downUp] )

Description

Performs an oversize or undersize operation on the layer you specify. This rule merges shapes that grow into one another and truncates acute angles.

Shapes that abut cannot pull apart in a negative sizing operation because the layer has been previously merged, unless you specified the unmergedIn option. (As of version 16, the [unmerged|unmergedIn|unmergedOut] option is not supported.) Similarly, this rule removes overlapping edges of shapes that were separate originally and extend into each other during a positive sizing operation. If you specify the unmergedOut option, the overlapping edges of the polygons remain separate.

Polygons that change as a result of geomSize maintain their original shape except when they have features that are less than or equal to twice the size value. In positive sizing, holes and notches may disappear, and angles may be truncated. In negative sizing, narrow sections and small corners may disappear.

Arguments

outLayer

Output layer.

inLayer

Input layer. Can be a derived polygon, edge or drawn layer.

size

Integer or floating-point number that defines the dimension of the size operation. This value can be negative or positive, but should be greater than 0.0005.

edges

Sizes each edge of a polygon or edges layer based on the size value.

Each edge is expanded into a rectangle where the size value is the dimension perpendicular to the edge. Positive sizing extends rectangles to the outside of the original shape, negative sizing extends rectangles to the inside of the original shape. If the resultant shapes overlap they are not merged.

This is the default mode for input edge layers.

downUp

Use with undersizing to eliminate small geometries that disappear after the undersizing. This options speeds checking when you need to use a large checking dimension in a drc check. You can also insert a downUp rule before the actual undersizing to speed up the sizing step.
geomStamp

```c
geomStamp(inLayer connectLayer [buttOrOver]
)
```

**Description**

Allows you to transfer electrical connectivity information (“net identifier”) from a previously connected layer to any other layer.

`geomStamp` assigns the net identifier of each `connectLayer` shape to the net identifier of each `inLayer` shape that the `connectLayer` shape overlaps. If an `inLayer` shape does not overlap any `connectLayer` shape, `geomStamp` assigns a new net identifier to the `inLayer` shape.

Like `geomConnect`, the `geomStamp` function always assigns a net identifier to an `inLayer` shape; it does not use an output layer.

If an `inLayer` shape overlaps more than one `connectLayer` shape, `geomStamp` counts the number of `connectLayer` shapes and transfers the net identifier associated with the majority of `connectLayer` shapes to the `inLayer`.

As of version 16, the `multStamp`, `floating`, `multConnect`, and `error` options are not supported.
geomStraddle

[outLayer =] geomStraddle( inLayer1 inLayer2 [limits] )

Description

Selects polygons on the first layer that straddle polygons on the second layer. A polygon straddles another if part of the layer1 polygon covers, abuts or is coincident to a layer2 polygon.

As of version 16, the connection option is not supported.

Arguments

- **outLayer**: Output layer.
- **inLayer1, inLayer2**: Input layers. Two layers are required.
- **limit**: Specifies the number of straddled polygons. If you use a limit, at least one shape must meet the criteria to be selected.

Specify the value by using a limit or range containing one of the following keywords:

- **keep**: Selects a shape if it adheres to the limit or range criteria.
- **ignore**: Rejects a shape if it adheres to the limit range criteria.

You can use these standard comparison operators with a limit or range:

- `<`
- `<=`
- `>`
- `>=`
- `==`
geomWidth

[outLayer =] geomWidth( inLayer keep { > | >= } width )

Description

Filters out areas with opposite edges whose distances are smaller than or equal to a specified width.

Warning:

This rule can in some cases alter the absolute angle of an edge. Because coordinates are represented by integers, coordinates that involve fractions or irrational numbers are rounded off, which can sometimes introduces a rotation of the sized edge. However, vertical and horizontal angles, and 45-degree angles that connect to verticals and horizontals, are maintained.

Arguments

- **outLayer**: Output layer.
- **inLayer**: Input layer containing the geometries to be extracted.
- **width**: The width value that will filter shapes. `geomWidth` will not select shapes smaller than or equal to this width.
layerDefs

layerDefs( format inlayer1 {inlayer2 inlayer3...} )

Description

This keyword starts a layerdefs section in the rules file. The layerdefs section defines original layers in your design. You can specify one or more sets of layerDefs in a single rules file.

Arguments

format Specifies the design format. You must include the quotation marks. “GDSII” is the only option L-Edit accepts.

inlayer1, inlayer2 ... User-defined layer names.

Unsupported Arguments

“df2”

Example

layerDefs( "GDS2"
  inlayer1
  inlayer2
  ...
  )


layerName

layerName = layer( layerId [type( dataType )] )

Description

Defines the original layers within a layerDefs section using a symbol name assigned to a layer
definition. You use this layerName in subsequent layer-derivation rules.

Arguments

-layerName : Output layer name.
-layer : Keyword that indicates the input layer is a geometric layer.
-layerId : For GDSII geometric layers, this is the layer number.

Layer names are case sensitive. To map multiple layers to a single
layerName, list the layers separated by spaces.

Layer names are restricted to 255 characters, the first character
of which cannot be a digit. Allowed characters are A–Z, a–z,
0–9, underscore (_), and question mark (?).

Layer names can contain special characters but they must be preceded
by a backslash (\). For example, n\>. Blank, tab, and paragraph
characters are not allowed.

dataType : dataType for the layer.

Example

metal2 = layer( 10 type( 11:13 ) )

This example combines the contents of the GDSII format layers 10, 11, 12, and 13 into a single metal2
layer with potentially overlapping polygons. These overlaps are removed automatically if the resultant
layer is used in subsequent operations.
offGrid (DRC Offgrid Data Check)

```
offGrid( inLayer gridSize [markerSize] [raw] ["message"] )
```

**Description**

The `offGrid` rule lets you evaluate a layout for grid errors by checking the top-level cell for offgrid data. Hierarchical information is accounted for so it reports errors are reported as if flat.

You can use `offGrid` to report separate and multiple output layers with one .

**Arguments**

- **offGrid**
  - Rule to check for offgrid coordinates.

- **inLayer**
  - Layer to be checked for offgrid coordinates. Can be an original layer or a derived layer.

- **gridSize**
  - Grid size against which the layer is checked for offgrid coordinates.

- **markerSize**
  - Value that indicates the length of the marker that indicates an offgrid vertex. If not specified, the default value is 1 user unit.

- **raw**
  - This option prohibits merging shapes on an original graphics input layer. If raw is not specified, the original layers will be merged before offgrid checking. It has no effect on derived layers.

- **“message”**
  - User-specified message that will be displayed in the error navigator. If not specified, the complete rule will be used as the message.

**Note:** Do not enclose the message in quotation marks.
**pinLayer**

```plaintext
layerName = pinLayer( "layerName" type( "purpose") )
```

**Description**

Use `pinLayer` to identify the shapes to be processed as pins. See also `pinText` (page 899) and `textToPin` (page 901).

**Arguments**

<table>
<thead>
<tr>
<th><code>layerName</code></th>
<th>Output layer name representing the pin layer.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pinLayer</code></td>
<td>Keyword instructing the software to find the geometry on the specified layer.</td>
</tr>
<tr>
<td>- <code>layerName</code>—the layer name, which must be enclosed in quotation marks.</td>
<td></td>
</tr>
<tr>
<td>- <code>type</code>—the type keyword identifies the layer-purpose name.</td>
<td></td>
</tr>
<tr>
<td>- <code>purpose</code>—enter the layer purpose, which must be enclosed in quotation marks.</td>
<td></td>
</tr>
</tbody>
</table>
pinText

\[
\text{layerName} = \text{pinText}(\text{layer-definition...})
\]

Description

Use \textit{pinText} to identify the text to be processed as pin names. \textit{pinText} creates labels from pins that are defined on a specified source layer.

You can name pins as geometric shapes on layers that become part of the design interconnect. The \textit{pinText} rule transforms the pins into a standard form that the software uses. See also \textit{textToPin} (page 901) and \textit{textToPin} (page 901).

\textbf{Note:} \textit{pinText} does not apply to GDSII. See \textit{textToPin} on page 901.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layerName</td>
<td>Output layer name representing the layer text.</td>
</tr>
<tr>
<td>pinText</td>
<td>Keyword instructing the software to find text objects in the specified layer.</td>
</tr>
</tbody>
</table>
| layer_definition | Use the following syntax: \[
layerId \ [type\{\text{dataType}\}] \]
|                | \begin{itemize}
|                | \item \textit{layerId}—the layer name, which must be enclosed in quotation marks.             |
|                | \item \textit{type}—the type keyword identifies the layer-purpose name, which must be enclosed in quotation marks. (If you do not specify the \textit{type} keyword, the default specifies all layer purposes except boundary.) |
|                | \item \textit{dataType}—specifies the layer-purpose name, which must be enclosed in quotation marks. |
|                | \end{itemize}                                                                               |
saveDerived

saveDerived( "message")

Description

Defines the contents of a derived layer as errors and writes the user-defined string to the Verification Error Navigator

See also offGrid (DRC Offgrid Data Check) on page 897.
textToPin

\[ \text{layerName} = \text{textToPin}\left( \text{layer\_definition}... \right) \]

**Description**

textToPin creates and names pins from an existing layout. You can name pins as geometric shapes on layers that become part of the design interconnect. The textToPin rule transforms the pins into a standard form that the software uses. See also **pinText** (page 899) and textToPin (page 901).

**Note:** Flattened text is not handled as pins, but labels are attached to the net if there is no conflict with other labels.

**Arguments**

- **layerName**
  
  Output layer name representing the layer text.

- **pinText**

  Keyword instructing the software to find text objects in the specified layer.

- **layer_definition**

  Use the following syntax:

  \[ \text{layerId} \ [\text{type(\ dataType)}] \]

  - **layerId**—the layer name, which must be enclosed in quotation marks.
  - **type**—the type keyword identifies the layer-purpose name, which must be enclosed in quotation marks. (If you do not specify the type keyword, the default specifies all layer purposes except boundary.)
  - **dataType**—specifies the layer-purpose name, which must be enclosed in quotation marks.
Tanner Commands with no Assura Equivalent

DRC check modifiers:

- edge, edgea, edgeb
- fig, figa, figb

Commands:

- extractCAP
- geonBkgnd
- geomConnect
- geomStamp
- layerName
- pinLayer
- saveDerived

Unsupported Assura Commands

L-Edit does not support all of the Assura commands, options or flags, particularly if they do not relate to DRC. Furthermore, L-Edit does not support

- DRC antenna checks
- DRC off-grid data checks
- Data Integrity checks
- Device Extraction rules
- Device parameter extraction rules and SKILL functions
- LVS comparison rules

Unsupported DRC check keywords:

- directional
- encBasedOvlp
- notch

Unsupported DRC check modifiers:

- boundary
- oppositeA, oppositeB
- predefined
- stepSize
- windowSize

Unsupported Assura commands

- geomCoinOnly
- geomEmpty
- geomGetAcuteAngle
- geomGetAdjacentEdge
- geomGetCorner, geomStretchCorner,
- geomGetCoverage
- geomGrowCorner
- geomLineEnd
- geomNodeRelate
- geomNot
- geomSizeAnd
- geomSizeAndNot
- geomSnap
- geomStretch
- geomStretchCorner
- geomTextShape
- geomXor
- keepLayer
- layer
- layerDefs2
- outfile
- pinTextFile
- text
- textFile
Layout vs. Layout (LVL) compares two L-Edit layout databases for their differences. It looks for moved, added and deleted geometry.

Typically you compare two different layout files to discover their possible differences. You can also compare two different cells in the same layout file. In either case, you can choose which layers L-Edit will examine. Invoke **File > Layout vs Layout Comparison** to perform Layout versus Layout.

There are two components to the LVL comparison results:

- A log file which shows the layers being compared and uses Boolean operators to detail the differences between the files, by layer for the layers with changes.
- An **LVL_Results** TDB file (which is temporary unless saved). This TDB file contains a cell for each of the input files, with generated layers containing objects that shows where and how the files differ for the Boolean conditions—**A NOT B** and **NOT A AND B**—considered. The file layers includes only the those selected for LVL comparison and the resulting generated layers.

### Select Cells and Layers to Compare

You can compare two cells in the same database, or two cells in two different databases. If cells in two separate databases are being compared, both databases must be open in L-Edit.
You can compare a single layer or multiple layers, in any combination.

The **Layers** checklist includes layers from both of the cells being compared.

In the **Layers** checklist, L-Edit displays layer names in pink if the top cell does not have geometry on the layer but the bottom cell does. (The layer is used in the B cell but not the A cell—A [not exist] B [Drawn/Special/In Use].)

In the **Layers** checklist, L-Edit displays layer names in red if the top cell has geometry on the layer but the bottom cell does not. (The layer is used in the A cell but not the B cell—A [Drawn/Special/In Use] B [not exist].)

The option “**Just create the derived layers but don’t generate the LVL results**” creates all the layers that are derived during LVL but does not draw the related layout. This option allows you to setup the layer derivation and then generate the layers as a separate step using **Tools > Add-Ins > Legacy Layer Generation** (see . It creates a larger number of derived layers than the full LVL process because during LVL any derived layers with no objects are deleted.

**View LVL Results**

When layout versus layout comparison is complete, L-Edit opens a log file and a temporary design file **LVL_Results LVL [File A_FileB.tdb]**.
LVL Log File

- **Layout vs. Layout Log** — The log header shows the L-Edit version used for LVL, the date and time it was performed, and the file location, name and cells compared.

- **Layers being compared** — This section of the log lists the layers selected for comparison.

- **Layers with changes** — Details the generated layers containing objects that show the differences between the input file layers compared.

- **Result A NOT B** indicates an object present in Cell A but not Cell B.

- **Result NOT A AND B** indicates an object not present in Cell A but present in Cell B.
The **LVL_Results** file contains only the drawn layers from the input files that were selected for comparison and the derived layers created by the LVL operation.

In the Layer palette for the **LVL_Results** file, layers that do not have a difference result are hidden (shown as grayed out in the palette) so that only the derived layers with objects resulting from differences, and the drawn layers where the differences occurred, will be displayed.

The **LVL_Results** file has one cell for each of the input files, with difference results placed as polygons on the appropriate generated layers in a third **LVL_Results** cell.
Layout vs. Layout Example

[1] Cell A: new-lights is compared to Cell B: new-lights_1, both from the same TDB file

Cell A: new-lights

Cell B: new-lights_1 is the same cell with certain geometry moved, added or removed.

[2] Cell LVL_Results LVLnew-lights_new-lights_1 contains the change objects.
The log shows that layers Active, P Select, Via, Metal1 and Active Contact were compared, and that eight derived layers were generated showing differences between the two cells.

Layers being compared:
- Active
- P Select
- Via
- Metal1
- Active Contact

Layers with changes:
- Layer: Result A NOT B - new-lights_Active - new-lights_1_Active
  - Layer A: new-lights_Active
  - Layer B: new-lights_1_Active
- Layer: Result A AND B - new-lights_P Select - new-lights_1_P Select
  - Layer A: new-lights_P Select
  - Layer B: new-lights_1_P Select
- Layer: Result A NOT B - new-lights_Metal1 - new-lights_1_Metal1
  - Layer A: new-lights_Metal1
  - Layer B: new-lights_1_Metal1
- Layer: Result A NOT B - new-lights_Active Contact - new-lights_1_Active Contact
  - Layer A: new-lights_Active Contact
  - Layer B: new-lights_1_Active Contact

We will examine the generated layers one by one. Note the layer patterns shown in the palette.

Right-click on a layer to open a menu where you can hide all but the selected layer.
With all other layers hidden it is easy to see the objects showing differences. Here the two rectangles on layer Active (green) show the results of A NOT B—objects that are present in Cell A but not Cell B.

The results for NOT A AND B on layer Active (green)—objects that are not present in Cell A but are in Cell B—are nil.
However, the results for NOT A AND B on layer P Select (pink checked) do show objects that are not present in Cell A but are in Cell B.
Extracting Layout

Extraction is a method of verifying a layout. The extraction process produces a netlist that describes the circuit represented by the layout in terms of device and connectivity information.

The extraction process is defined by making associations between patterns of layout geometry and the circuit components they represent. These associations are defined in the extract definition file.

The L-Edit general device extractor:

- Recognizes active devices (BJTs, diodes, GaAsFETs, JFETs, and MOSFETs), passive devices (capacitors, inductors, and resistors), and non-standard or compound devices by means of subcircuit recognition.
- Maintains process independence by means of an extract definition file, which describes how layers interact electrically.
- Uses device definitions that can be specified using generated layers, for a greatly expanded set of possible definitions. Derived layers are generated and disposed of automatically.
- Works with the most common device parameters, including resistance, capacitance, and device length, width, and area. These parameters provide useful information when verifying drive, fanout, and other circuit performance characteristics.
- Allows for 45-degree and all-angle geometry and, with HiPer Verify, hierarchical netlist extraction.
- Creates a netlist file in Berkeley 2G6 SPICE format, usable with any tool that reads a SPICE netlist. This netlist is ideal for use with the Tanner T-Spice circuit simulator to verify device sizes, drive capabilities, and other circuit performance factors, the LVS netlist comparator to check the equivalence of netlists generated from different sources, and the Tanner HiPer PX parasitic extractor.

The “Configuration Example” on page 921 illustrates the concepts described in this section.

Extract Definition Files

The extract definition file contains a list of the connect statements and devices to be extracted. This file can be used to define:

- Connections between two different process layers
- Devices definitions in terms of type, component layers, pins, and model names

See the section “Extract Definition File Format” on page 924 for further details on extract definition files syntax. L-Edit is shipped with a set of extract definition files that correspond to various technology processes. You can modify these files as necessary to define additional connections and devices for extraction.
Configuring the Extractor

To run Extract you must first load and select the extract definition files you want to run, then use **Tools > Extract** to perform the netlist extraction from the active cell using the extract rule(s) you have specified in the **Setup Extract** dialog.

Note that a netlist will not be exported if a design rule error exists.

The L-Edit extract engine can process Tanner .ext, Calibre .cal or Dracula .dra extract files. Note that Tanner .ext files do not contain layer derivations; those are in the .tdb file instead.

Select **Tools > Extract Setup** from the menu or use the **Setup Extract** button ( ) on the verification toolbar to open the **Setup Extract** dialog, which has a **General** and an **Options** tab.

**Setup Extract—General**

In addition to loading and selecting extract definition files, you also use **Setup Extract** to set the order in which rule checks are executed and as a shortcut to open a text editor where you can edit the rule sets.

---

See “Setup Extract Control Buttons” on page 915 for a description of these controls.
Chapter 24: Extracting Layout  Configuring the Extractor

**Extract Rule Sets to run**
Lists the Extract command files that are loaded and available to run. Only those rule sets that have their checkboxes in the “checked” state will be run when Extract is invoked.

Rule sets are run in the order in which they appear in this list.

All other functions in the dialog are performed only on the rule set that is highlighted.

**Calibre Defines**
Use this field to enter variable or variable and value combination to trigger preprocessor commands that are written in the rule files using `#DEFINE` and `#IFDEF`.

*Note:* Extract setup values in this field are saved separately from DRC define values.

**SPICE extract output file**
Browse to or enter the name of the SPICE netlist generated by the selected extractor.

**Overwrite existing output file**
Check this box to overwrite any existing output files of the same name without a confirmation.

**Open SPICE output file after extracting**
Check this box to open the SPICE extract file after the extract run.

**Send E-mail to:**
Enter an mail address to send notification to the specified recipient when the Extract job is complete. Use Configure to set the E-mail options shown below. Note that most E-mail applications will require a response prior to sending an E-mail initiated from another application.

**Configure Mail Service**

- **MAPI**
- **SMTP**

**SMTP Settings**
- **SMTP server:**
- **Mail from:**

**Test recipient:**

**Play sound**
Check to play a sound when Extract is complete. Press Configure to access standard Windows controls using the Sounds and Audio Devices Properties dialog.
Pop up message box

Check to open an Extract Completion Report when Extract is complete.

Only for warnings

Opens a message box only if L-Edit finds an error that generates a warning.

Setup Extract Control Buttons

Edit selected command file

Opens the selected rule set for editing. If Extract Standard Rule Set is highlighted, the Setup Extract Standard Rule Set dialog opens. If a command file is selected, that file opens in an L-Edit text window.

Add command file to list

Press this button to add a new entry to the list of command files. Use the browse ( ) button to find and select a file.

Delete command file from list

Removes the highlighted file from the list.

Move Up

Moves the highlighted file up in the list.

Move Down

Moves the highlighted file down in the list.
Setup Extract—Options

- **Statements to include before netlist**: Text entered in this field is added verbatim to the netlist. For example, enter a `.include` command to add a model or device name.
- **Write empty subcircuit definitions**: When checked, empty subcircuit definitions will be written to the netlist. (See “Using a “SUBCKT” Device to Extract Non-Standard Devices” on page 924.)
- **Write .model statements**: When checked, model definitions will be written to the netlist.
- **Save node-highlighting data**: When checked, node highlighting data is saved with the .tdb file. Use the Save Folder button to select the directory where the data is saved.

**Note**: The Setup DRC dialog and the Setup > Design—Node Highlighting dialog each include the option to save node highlighting data and to choose where it is saved. Be careful to manage these settings to avoid losing or overwriting your data.
Chapter 24: Extracting Layout

**Place ports as device labels**

When **Place ports as device labels** is checked and **To layer** is selected, the extract command tries to place recognized ports by reading the information from the SPICE file that was just created by the extractor. When L-Edit places labels in this way, it deletes and replaces the old labels.

You can also use the **Command** button in the **Verification Error Navigator** to select this option.

![Diagram of verification error navigator](Image)

**To layer**

Select the layer on which port names will be written.

**Write hierarchical netlist**

When checked, includes design hierarchy information in the netlist.

In the comments below the netlist for each subcircuit, L-Edit writes each device name as it appears in the netlist, then in parenthesis writes the model name, then the number of that device or model that appears in the subcircuit. Only non-zero counts are written, and the count includes primitive devices.

The total number of devices and the total number of nodes is written at the bottom of the netlist. When computing the node count for a subcircuit, nodes that are connected to subcircuits are counted as well as nodes that are connected to primitive devices.

When writing a hierarchical netlist, the total count is of the cumulative top level and subcircuit device counts—in other words, the sum of the toplevel and subcircuit counts. When writing a flat netlist, the count is the true flattened count of all devices.
Chapter 24: Extracting Layout Devices and Connections

The first step is to determine the specific classes of devices and connections that are to be extracted.

- A **device** is any circuit element (transistor, resistor, capacitor, diode, etc.).
- A **connection** is any electrical connectivity between two process layers, such as between the Poly and Metal1 layers when a contact is present on the Poly Contact layer.

Only relevant devices and connections need be defined. For example, every design contains resistors, because no process layer is a perfect conductor. But if the design to be extracted does not contain any

### Write node name aliases

When checked, writes node name aliases to the netlist. This option is only available when writing a flat netlist – “Write hierarchical netlist” must be turned off for “Write node name aliases” to be active.

As shown below, the name used in the netlist is listed first, with the aliases not used in the netlist listed after.

```
*************** Net name aliases report ***************
*  * Aliases of net Core/N_13
*    Core/U6/X1:X7/Data
*    Core/U6/X1:X7/QB
*    Core/U7/X1:X14/B
*
* Aliases of net Core/N_14
*    Core/U5/X1:X16/B
*    Core/U6/X1:X12/Out
* *
* Aliases of net Core/N_15
*    Core/U6/X1:X13/Out2
* *
* Aliases of net Core/N_16
*    Core/U5/X1:X8/Out
* *
* Aliases of net Core/Vdd
*    Core/U6/X1:X15/Vdd
*    Core/U7/X1:X10/Vdd
*    Frame/XL1/Vdd
*    Frame/XL2/Vdd
*    Frame/XR3/Vdd
```

### Run in background

Check this box to run Extract in background. Background processing means that you can continue to edit and perform other L-Edit operations while Extract is running. Extract results are returned as soon as they are found, so you can browse and correct errors before the entire Extract job is complete.

### Lock participating cells while Extract runs

Locks the cell, and all hierarchy below that cell, to prevent edits while Extract is running.

### Extract as:

- **Top level cell**
  - Writes the netlist as the top level of a design.
- **Subcircuit**
  - Writes the netlist as a subcircuit.

### Devices and Connections

The first step is to determine the specific classes of devices and connections that are to be extracted.

- A **device** is any circuit element (transistor, resistor, capacitor, diode, etc.).
- A **connection** is any electrical connectivity between two process layers, such as between the Poly and Metal1 layers when a contact is present on the Poly Contact layer.

Only relevant devices and connections need be defined. For example, every design contains resistors, because no process layer is a perfect conductor. But if the design to be extracted does not contain any
wire long enough for its inherent resistance to affect the circuit’s performance, then the wires do not have to be defined and extracted as resistors.

**Finding Devices and Nodes**

**Tools > Goto Device (Alt+G)** locates a device or node from the extract output file by zooming to it and placing a marker in the layout. You must have extracted connectivity first or pointed to the correct SPICE file for this operation to function. The behavior is slightly different for the two search types.

You can toggle the marker display on ( ) and off ( ) with the lightbulb toolbar button and delete all markers with the eraser toolbar button ( ).

**Tools > Goto Device**

**GoTo**

Enter the **Name** of the device or node you want to locate.

**Look for**

Select **Device** or **Node**.

**Note:** For nodes to appear in the extract output file, you must check the **Write node-element cross-reference** option in the **Output** tab of the **Tools > Extract** dialog.

**Mouse Pick**

Click this button to select a device with a mouse click in the layout.
When you enter a device, L-Edit will zoom to it as shown below.

When you enter a node, L-Edit will highlight the elements connected to that node and create disposable ports on the error layer that you can reposition to better inspect the node.

**Extracting Resistor and Capacitors**

To extract resistors and capacitors, you must in addition enter the following three constants for each involved layer. Use "General Layer Parameters" (page 91) to enter these values.

- An *area capacitance* (in attofarads per square micron)
- A *fringe capacitance* (in femtofarads per micron)
- A *resistivity* (in ohms per unit area)

*Capacitance* is the sum of two products: that of the area of the capacitor and the area capacitance, and that of the perimeter of the capacitor and the fringe capacitance. (Capacitors are polygons on the recognition layer.) *Resistance* is the product of the resistivity and the length of the resistor, divided by the width.
Wire Conversions in Layer Generation

Wires involved in layer-generation operations on the source layers are converted to polygons on the generated layer. However, some join styles on 45° wires result in wire edges that meet at non-45° angles. To ensure that the resulting objects are true 45° polygons when the layer is generated, the problem joins are modified. Round joins are processed as layout joins and round ends are processed as butt ends to satisfy the 45° polygon criteria.

Examples of a converted 45° source wire to a generated layer polygon are shown in the following figure:

After rescaling, the finer grid resolution (more internal units per technology unit) allows the generated polygon to approximate the source wire more closely. Without rescaling the design, the vertex of the generated polygon in the above example is forced to the nearest internal unit that maintains a 45° angle.

Configuration Example

The following example illustrates how to configure the extractor to recognize a transistor in a CMOS n-well process. It shows how transistors may be clearly and uniquely identified by generated layer definitions and device statements in the extractor definition file. Other SPICE devices may be identified in similar fashion.

An NMOS transistor in an n-well CMOS process consists of:

- The channel
- A source pin of n-doped diffusion material touching the channel
- A gate pin of polysilicon over the channel
- A drain pin of n-doped diffusion material touching the channel
- A bulk pin to the substrate
When the extractor finds a configuration of polygons in the layout corresponding to this definition, it should write an NMOS transistor statement into the output file (netlist).

**Device Definition**

The following statement causes a MOSFET to be generated in the output.

```
# NMOS transistor
device = MOSFET(
    RLAYER=ntran;
    Drain=ndiff, AREA, PERIMETER;
    Gate=poly wire;
    Source=ndiff, AREA, PERIMETER;
    Bulk=subs;
    MODEL=NMOS;
)
```

The recognition layer is defined as `ntran`, and the pin layers are defined as `ndiff`, `Poly`, and `subs`. This causes the extractor to recognize a MOSFET wherever it sees `ntran` geometry, touched by geometry on `ndiff`, `Poly`, and `subs`.

However, MOSFETs are not typically created by drawing geometry on `ntran`, `ndiff`, or `subs`. They are created by drawing `Poly` geometry over `Active` geometry inside `NSelect` geometry.

To generate layers to create the correct geometry on the `ntran`, `ndiff`, and `subs` layers from user-drawn geometry on `Active`, `Poly`, and `N Select` layers.

**Recognition Layers**

A transistor gate is formed on the chip when `Poly` geometry and `Active` geometry intersect on the layout. The generated layer

```
gate = ( Poly ) AND ( Active )
```

is used to define a generic transistor gate.

However, a CMOS process will have both NMOS transistors (in the substrate) and PMOS transistors. The `gate` layer definition does not differentiate between the two.

L-Edit uses a default CMOS setup that assumes a `p`-substrate, has a nongenerated layer (`N Well`) for defining the `n`-well, but does not have a layer for defining the substrate surface. The generated layer

```
subs = NOT ( N Well )
```

is used to define the substrate surface.

Now, two generated layers can uniquely identify NMOS and PMOS transistor channels:

```
ntran = ( gate ) AND ( subs )
ptran = ( gate ) AND ( N Well )
```

**Pin Layers**

When the extractor identifies a transistor to be written to the output netlist, it looks for the pins that should be touching the recognition layer if the device is properly constructed.
A MOSFET has four pins attached to it: drain, gate, source, and bulk. The gate is defined to be the Poly geometry that touches the transistor. The bulk in a PMOS device is the N Well, and in an NMOS device is the substrate (subs). For these pins, the proper layers are already defined.

In the layout, a single polygon on the Active layer stretches across the whole transistor, but in a fabricated chip, the diffusion material will not exist under the gate. The generated layer

\[ \text{Field Active} = (\text{Active}) \text{ AND } \lnot(\text{Poly}) \]

creates geometry on either side of, but not underneath, a transistor gate.

Finally, an NMOS transistor has source and drain pins made up of \(n\)-doped material, and a PMOS transistor has source and drain pins made of \(p\)-doped material. The doping type is controlled by drawing geometry on the N Select and P Select layers, so two generated layers can uniquely identify both pin layers:

\[
\begin{align*}
\text{ndiff} &= (\text{Field Active}) \text{ AND } (\text{N Select}) \\
\text{pdiff} &= (\text{Field Active}) \text{ AND } (\text{P Select})
\end{align*}
\]

### Detecting Soft Connections

You can configure the extractor to write special devices for each connection to a well or substrate, called “soft connections.”

These special devices must be included in the netlist for LVS to identify soft-connected nodes (see “Detecting Soft Connections with LVS” on page 987). By convention, these devices are 0\(\Omega\) resistors with designated model types, such as R\_WELLCONTACT and R\_SUBSCONTACT.

To write special devices for soft connections, make the following changes in the extract file:

- To detect ohmic contacts to well and substrate, create two additional derived layers:
  \[
  \text{ohmic well contact} := \text{n well wire AND ndiff} \\
  \text{ohmic substrate contact} := \text{subs AND pdiff}
  \]

- Do \text{not} connect wells and substrates to diffusions through ohmic contacts. Look through the extract file and delete lines like the following:
  
  connect(n well wire, ndiff, ndiff) \\
  connect(subs, pdiff, pdiff)

- Add device recognition statements for the ohmic contact special devices (resistors). For example:

  ```
  # Well contact
  device = RES(
    RLAYER=ohmic well contact;
    Plus=n well wire;
    Minus=ndiff;
    MODEL=WELLCONTACT;
  )

  # substrate contact
  device = RES(
    RLAYER=ohmic substrate contact;
    Plus=subs;
    Minus=pdiff;
    MODEL=SUBSCONTACT;
  )
  ```
Chapter 24: Extracting Layout

Using a “SUBCKT” Device to Extract Non-Standard Devices

You can use the SUBCKT device in the EXT file to extract non-standard devices such as resistors with a bulk connection. The directory L-Edit Pro\Samples\Extract\ICResistors contains an example demonstrating the extraction of a resistor with a bulk connection.

Here is a SPICE subcircuit that can model an IC resistor with a bulk connection.

```plaintext
.SUBCKT ICResPoly Plus Minus Bulk L=1 W=1
.PARAM Poly_Conduct = 140
.PARAM Rs='L/(W*Poly_Conduct)'
.PARAM Eox = '3.9*8.85E-12'
.PARAM Tox = 3.750E-7
.PARAM Cs='L*W*Eox/Tox'
R1 Plus 2 R='Rs/2'
R2 2 Minus R='Rs/2'
C1 Plus Bulk C='Cs/4'
C2 2 Bulk C='Cs/2'
C3 Minus Bulk C='Cs/4'
.ENDS
```

The subcircuit has three terminals and needs the length and width of the device. We can extract this using the subcircuit device in the EXT file.

```plaintext
# IC Poly Resistor
device = SUBCKT(
    RLayer=PolyResistor, LW;
    Plus=PolyWire, DEVICEWIDTH;
    Minus=PolyWire, DEVICEWIDTH;
    Bulk=Substrate;
    MODEL=ICResPoly;
)
```

This device definition uses the layer PolyResistor for the recognition layer, which is Poly & Resistor ID. This makes sure that we have to have both Poly and ID to have a resistor. If we used just the ID, then there could be a chance that we create a resistor with just the ID and not the Poly. Extract will think there is a resistor there when physically there is not. The positive and negative pins use PolyWire, which is Poly & NOT Resistor ID. If we used just Poly, the resistor would be shorted. The keyword DEVICEWIDTH is used on the positive and negative pins so that the width of those pins will be used to calculate the width of the device. The bulk connection use Substrate, which is NOT N Well.

When you run extract, make sure to check “Write empty subcircuits definitions” so you can .INCLUDE your subcircuit definition for the resistor by using the “SPICE include statement” option. The result will look like the following.

```plaintext
.INCLUDE ResSubs.sp
X1 1 2 4 ICResPoly L=2.4u W=720n
```

Extract Definition File Format

The .ext extract definition file contains a list of comments, connection statements, and device statements. L-Edit is shipped with a directory containing a set of extract definition files that correspond to various technology processes. You can modify these files as necessary to define additional connections and devices for extraction.

Extract definition files must conform to the following restrictions:
Layer names are case-sensitive, and must match the case of layer names defined in the TDB file. The rest of the definition file is case-insensitive; upper and lower cases can be used interchangeably.

Layer names cannot contain commas or semicolons and they cannot be longer than 40 characters.

Layer names cannot have leading or trailing spaces.

Pin names cannot contain commas, semicolons, or spaces, and they cannot be named MODEL.

Model names cannot contain commas, semicolons, spaces, or closing parentheses.

For compatibility with existing extract definition files, the WIDTH keyword is ignored for all devices except a GAASFET/MESFET.

IGNORE_SHORTS indicates that if the device has all of its pins connected to the same node then it will be considered shorted and the device will be written to the extract netlist file as a comment.

Comment Statements

A comment statement begins with a pound sign (#) and continues to the end of the line:

# This is an extract definition file comment.

Connect and Attach Statements

A connection statement defines a connection between two different layers. A connection always involves three layers: the two layers being connected and the layer through which they connect. Connection statements have the following format:

connect(Layer1, Layer2, ThroughLayer)

where Layer1 and Layer2 are the names of the layers being connected, and ThroughLayer is the name of the connecting layer. For example:

# Connect Poly to Metal1
CONNECT(Poly, Metal1, PolyContact)

The connect statement does an AND operation of Layer1 and ThroughLayer, and the resulting geometry has to either overlap or touch Layer2 in order for it to make a connection. Therefore, if Layer1 and ThroughLayer are not overlapping, no connection will be made. For example, if you wanted to connect N-Well to n diffusion you could write two separate connect statements.

# Connects N-Well to n diffusion.
# N-Well and n diffusion have to overlap to make a connection.
connect(NWell, nDiff, nDiff)

# Connects N-Well to n diffusion.
# N-Well and n diffusion have to overlap or touch to make a connection.
connect(nDiff, NWell, nDiff)

The Tanner .ext format also supports the attach command, which is used to label nets with names using ports on the layout. The attach command attaches a name from a port object on one layer to a net with a polygon on a second layer that overlaps the port. The second layer must appear as one of the layers in a connect statement.

The attach command has the following format:

attach(Layer1, Layer2)
where Layer1 is the layer containing the port object and Layer2 is the polygon layer to which the net name should be attached.

**Substrate Node Statement**

Nodal parasitic capacitances are referenced from the node to the substrate. You can designate the substrate node by indicating the substrate layer. Extract will then find the first node connected to the substrate layer and use that as the substrate node when writing the parasitic nodal capacitors. If no substrate layer is indicated, Extract will use node 0 or ground as the substrate node. The substrate node statement has the following format:

```
SUBSTRATE_NODE = SubstrateLayer;
```

where `SubstrateLayer` is the name of the substrate layer. For example:

```bash
# Use Subs as the Substrate node.
SUBSTRATE_NODE = subs;
```

results in the following SPICE output:

```
Cpar1 5 2 10f
```

where 2 is the substrate node.

**Device Statements—General Format**

A device statement defines a device. Passive (capacitors, resistors, and inductors,) active (BJTs, diodes, GaAsFETs, JFETs, MOSFETs), and subcircuit devices are specified with the same general format.

For all device statements, it is necessary to identify a recognition layer, which is the layer Extract uses to recognize the device. You may specify multiple devices with the same recognition layer as long as they have different pin configurations. This technique is particularly useful in extracting multisource/drain transistors. The recognition layer is defined as follows:

```
RLAYER = rLayer;
```

where `RLAYER` is required, and `rLayer` is the name of the recognition layer.

Following the recognition layer is a list of pins of the device. The order of this list determines the order of the pins in the extracted netlist. The extractor does not require any particular order, but LVS requires that both source netlists contain pins in the same order, and SPICE simulators also have strict rules about the order in which pins appear. We recommend following the standard SPICE orders:

- BJT devices: collector—base—emitter—substrate
- MOSFET, JFET, and GaAsFET/MESFET devices: drain—gate—source—bulk
- Diodes, resistors, capacitors, and inductor devices: positiveNode—NegativeNode

If the pin names used in the EXT file are **Collector**, **Base**, **Emitter**, and **Substrate** (BJT devices), or **Drain**, **Gate**, **Source**, and **Bulk** (all other active devices), they are sorted automatically in the default SPICE order.

Pins are specified as follows:
pinName = pinLayer;

where pinName is the name of the pin and pinLayer is the name of the associated layer.

A model definition follows the list of pins. This definition is not required for passive devices, where MODEL =; is acceptable. The model name, if present, will be written into the extracted netlist. For SPICE, model names are not required for capacitors, resistors, inductors, or diodes, but are required for all other devices.

For passive devices, model statements have the form:

MODEL = [model];

For active devices, model statements have the form:

MODEL = model;

where MODEL = is required and model is the optional model name. The empty statement MODEL =; is still required if no model name is specified.

Device Statements–Specific Formats

In the following format specifications:

- Unitalicized words and characters (except the bracket characters [ and ]) are to be entered as shown.
- Words and characters enclosed by brackets ([ ]) are optional.
- Words and characters enclosed by braces and separated by a vertical pipe—{option1 | option2}—represent alternates. You can use the syntax on the left or the right side of the pipe character.
- Variables containing the string Layer represent layer names.
- model represents the SPICE model name for the device.

Capacitor

DEVICE= CAP {
    RLayer = rLayer [{AREA} | {LW}];
    Plus = Layer1 [,AREA];
   Minus = Layer2 [,AREA];
    MODEL = [modelName ];
} [IGNORE_SHORTS]

A capacitor has the following format in the SPICE output statement:

AREA keyword
Cxxx n1 n2 [ModelName] [C=cValue]

LW keyword
Cxxx n1 n2 [ModelName] L=cLength W=cWidth

The following rules apply to capacitors:

- The optional AREA keyword for a capacitor may be specified on only one layer (recognition or pin layer) and is used to indicate the layer for which the capacitance will be calculated.
- If no AREA keyword or LW keyword is present, the capacitance will be based on the area of the recognition layer (rLayer).
- The LW keyword cannot be used with the AREA keyword.
- The LW keyword can only be used with the recognition layer (rLayer).
- Capacitance is calculated as follows:
  \[ C_{\text{total}} = C_{\text{area}} + C_{\text{fringe}} \]
  \[ C_{\text{area}} = (\text{Area of the Layer}) \times (\text{Layer's Area Capacitance}) \]
  \[ C_{\text{fringe}} = (\text{Perimeter of the Layer}) \times (\text{Layer's Fringe Capacitance}) \]
- The area capacitance (aF/sq. micron) and fringe capacitance (fF/micron) are specified in the Setup Layers dialog for each specific layer.
- Capacitor average length and width are calculated as if the capacitor was a rectangle. They are calculated as follows:
  \[ L_{\text{perimeter}} = \text{Perimeter of the Layer} \]
  \[ L_{\text{area}} = \text{Area of the Layer} \]
  \[ C_{\text{length}} = \frac{1}{4} \cdot L_{\text{perimeter}} + \frac{1}{4} \sqrt{L_{\text{perimeter}}^2 - 16 \cdot L_{\text{area}}} \]
  \[ C_{\text{width}} = \frac{1}{4} \cdot L_{\text{perimeter}} - \frac{1}{4} \sqrt{L_{\text{perimeter}}^2 - 16 \cdot L_{\text{area}}} \]

**Resistor**

DEVICE=RES(
  R_LAYER = rLayer [, LW];
  Plus = Layer1;
  Minus = Layer2;
  MODEL = [ModelName];
) [IGNORE SHORTS]

A resistor has the following format in the SPICE output statement:

Rxxx n1 n2 [ModelName] [R=]rValue

**LW keyword**
Rxxx n1 n2 [ModelName] L=rLength W=rWidth

The following rules apply to resistors:
- Resistance is calculated with the formula \( R = \rho \times (l/w) \), where \( \rho \) is the sheet resistance in units of ohms/square, \( l \) is the length of the resistor, and \( w \) is the width of the resistor.
- The value of \( \rho \) is taken from the number specified with the Setup Layers dialog for the recognition layer (rLayer) of the resistor.
- The LW keyword can only be used with the recognition layer (rLayer).
- The values of \( l \) and \( w \) are determined from the layout. The extractor computes the area of the recognition layer and divides it by the effective width to obtain \( l \). The width of a pin is the length of...
the edge that the pin shares with the recognition layer (rlayer). The effective width is the average of the plus pin width and minus pin width.

![Diagram showing plus pin width and minus pin width]

**Inductor**

DEVICE=IND (  
  RLayer = rLayer ;  
  Plus = Layer1 ;  
  Minus = Layer2 ;  
  MODEL = [model ] ;  
) [IGNORE_SHORTS]

An inductor has the following format in the SPICE output statement:

L name n1 n2 [model] [L=]

Extract does not calculate inductance; users must add this value after netlist extraction.

**BJT**

DEVICE=BJT (  
  RLayer = rLayer [,AREA];  
  Collector = cLayer [,AREA];  
  Base = bLayer [,AREA];  
  Emitter = eLayer [,AREA];  
  [Substrate = sLayer ]];  
  MODEL = model ;  
  [NominalArea = areaVal ;]  
) [IGNORE_SHORTS]

A BJT device has the following format in the SPICE output statement:

Q name col bas emt [sub] model [AREA= {rLayerArea | pinArea } /areaVal]

The following rules apply to BJT devices:

- The optional AREA keyword can be specified on only one layer (the recognition layer or the pin layer). It is used to indicate the layer for which the area is to be calculated.
- Nominal area can be expressed either in decimal or scientific notation. It has units of m², but no unit tag will appear after the value.
- The NominalArea keyword is required if the AREA keyword is used.
- If no AREA keyword is present, the area will not be written to the SPICE statement.
Diode

DEVICE=DIODE {
    RLayer = rLayer [, AREA];
    Plus = Layer1 [, AREA];
    Minus = Layer2 [, AREA];
    Model = model ;
    [NominalArea = areaVal ;]
} [IGNORE_SHORTS]

A diode has the following format in the SPICE output statement:

Dname n1 n2 model [AREA= {rLayerArea | pinArea } /areaVal]

The following rules apply to diodes:

- The optional AREA keyword can be specified on only one layer (the recognition layer or the pin layer). It is used to indicate the layer for which the area is to be calculated.
- Nominal area can be expressed either in decimal or scientific notation. It has units of m², but no unit tag will appear after the value.
- The NominalArea keyword is required if the AREA keyword is used.
- If no AREA keyword is present, the area will not be written to the SPICE statement.

GAASFET/MESFET 1

The following syntax can be used to define a GAASFET/MESFET device using its nominal area as a parameter.

DEVICE=GAASFET {
    RLayer = rLayer [, AREA];
    Drain = dLayer [, AREA];
    Gate = gLayer [, AREA];
    Source = sLayer [, AREA];
    [Bulk = [bLayer];]
    Model = model ;
    [NominalArea = areaVal;]
} [IGNORE_SHORTS]

The GAASFET/MESFET device has the following format in the SPICE output statement:

Zname drn gat src [blk] model [AREA= {rLayerArea | pinArea } /areaVal]

The following rules apply to GAASFET/MESFET devices:

- The optional AREA keyword can be specified on only one layer (the recognition layer or the pin layer). It is used to indicate the layer for which the area is to be calculated.
- Nominal area can be expressed either in decimal or scientific notation. It has units of m², but no unit tag will appear after the value.
- The NominalArea keyword is required if the AREA keyword is used.
- If no AREA keyword is present, the area will not be written to the SPICE statement.
- The GAASFET/MESFET 1 syntax is distinguished by the presence of the AREA, and/or NominalArea keywords. L-Edit determines the appropriate output based on the presence of either of these keywords.


**GAASFET/MESFET 2**

The following syntax can be used to define a GAASFET/MESFET device using its width as a parameter.

```
DEVICE=GAASFET (
    RLAYER = rLayer ;
    Drain = dLayer [, WIDTH];
    Gate = gLayer ;
    Source = sLayer [, WIDTH];
    [Bulk = [bLayer ];]
    MODEL = model ;
) [IGNORE_SHORTS]
```

A GAASFET/MESFET device has the following format in the SPICE output statement:

```
Zname drn gat src [blk] model L=length W=width
```

The following rules apply to GAASFET/MESFET devices:

- The length is the length of gate, and the width is the length of the edge that the indicated layer shares with the recognition layer \(rLayer\). The length and width have units of meters.
- The optional **WIDTH** keyword for a GAASFET/MESFET may be specified on only the drain or source pin but not both, and is used to indicate the layer for which width will be calculated.
- If no **WIDTH** keyword is present, the width and length will not be written to the SPICE statement.
- The GAASFET/MESFET 1 syntax is distinguished by the presence of the **WIDTH** keyword. L-Edit determines the appropriate output based on the presence of this keyword.

**JFET**

```
DEVICE=JFET (
    RLAYER = rLayer [, AREA];
    Drain = dLayer [, AREA];
    Gate = gLayer [, AREA];
    Source = sLayer [, AREA];
    [Bulk = [bLayer ];]
    MODEL = model ;
    [NominalArea = areaVal ;]
) [IGNORE_SHORTS]
```

A JFET device has the following format in the SPICE output statement:

```
Jname drn gat src [blk] model [AREA= {rLayerArea | pinArea } /areaVal]
```

The following rules apply to JFET devices:

- The optional **AREA** keyword can be specified on only one layer (the recognition layer or the pin layer). It is used to indicate the layer for which the area is to be calculated.
- Nominal area can be expressed either in decimal or scientific notation. It has units of m\(^2\), but no unit tag will appear after the value.
- The **NominalArea** keyword is required if the **AREA** keyword is used.
- If no **AREA** keyword is present, the area will not be written to the SPICE statement.
A MOSFET device has the following format in the SPICE output statement:

```
Mname drn gat src [blk] model L=lengthValue W=widthValue [{[AD=areaValue] [PD=perimeterValue] [AS=areaValue] [PS=perimeterValue] | [GEO=#]}
```

The following rules apply to MOSFET devices:

- The length is the length of gate, and the width is the average length of the edges that the source and drain share with the recognition layer (rLayer). The length and width have units of meter.
- The optional AREA keyword may be specified on the drain or source pin or both. It is used to indicate whether the area for that layer will be calculated and written for the AD (Area of the Drain) and AS (Area of the Source) output values.
- The optional PERIMETER keyword may be specified on the drain or source pin or both, and is used to indicate whether the perimeter for that layer will be calculated and written for the PD (Perimeter of the Drain) and PS (Perimeter of the Source) output values. The shared edge is not included in perimeter calculations for a source or drain.
- The optional /GATE=# keyword is used with the PERIMETER keyword. It may be specified on the drain or source pin or both, but only where the PERIMETER keyword has already been designated. The number is a decimal value between 0.0 and 1.0, indicating the fraction of the gate width to include in the perimeter. If the /GATE=# keyword is missing, the perimeter will include the gate width.
- The optional GEO keyword may be specified on only the drain pin. It is used to indicate that the GEO value will be written to the SPICE statement. The GEO keyword can only be used if the AREA and PERIMETER options are not used. The values of GEO written to the SPICE statement are as follows:
  - GEO=0—drain and source areas are not shared
  - GEO=1—drain is shared
  - GEO=2—source is shared
  - GEO=3—both drain and source are shared
- The GEO keyword is used with the area calculation method (ACM) for modeling the source and drain diodes. Users are encouraged to use the AREA and PERIMETER options because they yield more accurate approximations of the area and perimeter values than those achieved using the GEO option.

**Subcircuit**

Subcircuits can be defined explicitly for the extractor. This method of describing subcircuits is different from automatic subcircuit instance recognition (see “Using a “SUBCKT” Device to Extract Non-Standard Devices” on page 924).

```
DEVICE=SUBCKT {
```
A subcircuit has the following format in the SPICE output statement:

```
AREA   Keyword
Xzzz n1 [n2 ...] cName [AREA=rLayerArea/areaVal]
    [PERI=rLayerPerimeter]
    [L=cLength W=cWidth]
    [AREA_pin1Name=pin1Area/areaVal]
    [PERI_pin1Name=pin1Perimeter]
    [WIDTH_pin1Name=pin1Width]
    [AREA_pin2Name=pin2Area/areaVal]
    [PERI_pin2Name=pin2Perimeter]
    [WIDTH_pin2Name=pin2Width] ...
```

The following rules apply to subcircuits:

- Nets that are internal to a subcircuit are not promoted up to the subcircuit interface.
- The optional `AREA` keyword may be specified on one or more layers (recognition or pin layer). An area will be calculated for each indicated layer.
- The optional `PERIMETER` keyword may be specified on one or more layers (recognition or pin layer). A perimeter will be calculated for each indicated layer.
- The optional `WIDTH` keyword may be specified on one or more pin layers. A width of the pin layer edge that is coincidence with the recognition layer will be calculated for each indicated layer. The `WIDTH` keyword may not be used with the recognition layer.
- The `LW` keyword can be used with the `AREA` and/or `PERIMETER` keyword.
- The `LW` keyword can only be used with the recognition layer (`rLayer`).
- The `DEVICEWIDTH` keyword can only be used if the `LW` keyword is present and it can only be used with the pin layers. This affects the calculation of `L` and `W`, see below.
- Nominal area can be expressed either in decimal or scientific notation. It has units of m², but no unit tag will appear after the value.
- The `NominalArea` keyword is required if the `AREA` keyword is used.
- If no `AREA` keyword is present, the area will not be written to the SPICE statement.
If no \texttt{DEVICEWIDTH} keywords exist on any pin, the Subcircuit average length and width are calculated as if the Subcircuit recognition layer geometry was a rectangle. They are calculated as follows:

\begin{align*}
L_{\text{perimeter}} & = \text{Perimeter of the Layer} \\
L_{\text{area}} & = \text{Area of the Layer} \\
C_{\text{length}} & = \frac{1}{4} \cdot L_{\text{perimeter}} + \frac{1}{4} \sqrt{L_{\text{perimeter}}^2 - 16 \cdot L_{\text{area}}} \\
C_{\text{width}} & = \frac{1}{4} \cdot L_{\text{perimeter}} - \frac{1}{4} \sqrt{L_{\text{perimeter}}^2 - 16 \cdot L_{\text{area}}} \\
\end{align*}

If one or more \texttt{DEVICEWIDTH} keywords exists on the pins, the Subcircuit average width is equal to the average width of the pins that are marked with \texttt{DEVICEWIDTH}. The length is equal to the \texttt{AREA/Width}. The width of a pin is the length of the shared edge between polygons on the pin layer and polygons of the recognition layer. An example is shown below.

\begin{verbatim}
DEVICE=SUBCKT ( 
  RLayer = Recognition Layer, LW; 
  pin1 = pin1Layer, DEVICEWIDTH; 
  pin2 = pin2Layer, DEVICEWIDTH; 
  pin3 = pin3Layer; 
  MODEL = MySub; 
)
\end{verbatim}

\begin{align*}
Pin1_{\text{width}} & = \text{Width of Pin 1} \\
Pin2_{\text{width}} & = \text{Width of Pin 2} \\
RL_{\text{area}} & = \text{Area of the Recognition Layer} \\
S_{\text{width}} & = \frac{Pin1_{\text{width}} + Pin2_{\text{width}}}{2} \\
S_{\text{length}} & = \frac{RL_{\text{area}}}{S_{\text{width}}} \\
\end{align*}
25 HiPer Parasitic Extract (PX)

With shrinking process technologies, the resistance and capacitance effects of interconnect wiring can have a significant impact on circuit performance, signal delay and signal noise. Tanner EDA’s parasitic extraction tool HiPer PX calculates the implicit resistance, ground capacitance and lateral capacitance of the required wiring in a layout to create an accurate analog model.

Running PX

Interconnect capacitance is calculated by giving the extraction tool the top view layout of a design in the form of input polygons on a set of layers; a mapping to a set of devices and pins (from a Layout Versus Schematic run), and a cross sectional understanding of these layers.

This information is used to create a set of layout wires that have added capacitors where the input polygons and cross sectional structure indicate. The output netlist, which is flat, contains the same set of input nets as the input design netlist and adds parasitic capacitor devices between these nets.

HiPer PX can’t represent layers easily and only need information about interconnect layers, some of which might be merged. As with other L-Edit processing, a notched polygon is considered a hole.

Because output terminals in hierarchical output all have netname vdo, terminal names are assigned unique device:terminal names. You can probe from S-Edit to see the results.

Choose a PX File

To avoid double-counting pins belonging to devices that are recognized by pin, you run HiPer Verify first to tag device pins with a name and location so the parasitic extractor can recognize them.

HiPer PX writes out transistors and parasitic behavior as a SPICE file where the output is total net capacitance plus the resistance between two pins. Note that a netlist will not be exported if a design rule error exists.

To run HiPer PX you must specify at least one parasitic definition (.px) file in the Setup Extract > General dialog. To open it, select Tools > Extract Setup from the menu or use the Setup Extract button ( ) on the verification toolbar.
Use the ( ) button to add your .px files to the **Extract Rule Sets to run** list.

**Choose Your PX Options**

[Image of Setup Extract window with options for capacitance, resistance, and circuit reduction]

[Image of Setup Extract window with option to complete PX files]
### Capacitance

Sets the mode and level of detail of capacitance information that will be extracted. Select an option from the drop-down menu:

- No extraction
- 2D substrate only
- 2D substrate and vertical
- 2D substrate, vertical and lateral
- 3D substrate only
- 3D substrate, vertical and lateral

The 2D solver uses table-based interpolation. The 3D solver uses physical information to construct a three-dimensional model of the circuit for computation. As a result it requires much more processing time so typically is used to characterize relatively small but very important circuit blocks.

Substrate mode gives you at most one capacitor per net, always going to ground. Lateral mode calculates parasitic effects within a layer. Vertical mode includes crosstalk between layers as well.

### Resistance

Check this box to calculate the average resistivity for a uniform sheet thickness, to the threshold value entered in the **Sheet resistance threshold** field, where the default value is 1.0. You can add `-Slow_sheet_res=x` to the command line as well.

The threshold instructs HiPerPX to treat good conductors as perfect conductors. Any layer whose sheet resistance is below the threshold will be treated as if its sheet resistance is 0. This speeds up extract and simplifies the resulting netlist, but will fail to account for some parasitic resistance that might otherwise have been included (such as extremely long, thin metal traces).

### Circuit reduction

Circuit reduction collapses series of resistances to fewer resistors equivalent to the original network. If you select **Frequency dependent**, you must enter a maximum hertz value. The other options are **Frequency independent** or **No reduction**.

### Compile PX files

Using initial information from the foundry-provided GIT tech file, converts and compiles a SPACE.DEF.T file from a SPACE.DEF.S file, a mask file and a default lambda file. (The default lambda file specifies the default unit distance in your layout, where for example, a value of 0.20 means that a distance of 1 in the layout editor corresponds to 0.2 microns.) See “PX File Format,” below.

HiPer PX will warn and offer to compile if space.def.t is missing or older than space.def.s

### PX File Format

The .px file contains various parameters for parasitic extraction, with locations, in one flat netlist as shown below.
layer PolyWire
layer pdiff
layer ndiff
layer Meta1
layer Meta2
layer Meta3
layer Meta4
layer Vial
layer Via2
layer Via3NoCap
layer NWellWire
layer Contact
space_tech_path scmos_n
svrf_deck Generic_250nm_LVS.cal
space_custom_args -pmax

You can use keywords with tecc_args {args} to override the default tecc arguments.

**layer**

Specifies the layers from the Calibre deck to be exported to PX. More than one layer statement is typical.

**space_tech_path**

Gives the absolute path name or path relative to this file of the location of the space runset files space.def.s and space.def.p that define the explicit device parameters, and space.def.t which Tanner generates for interconnects, SVRF layer names (without purposes) and layer numbers.

**svrf_deck**

Gives the location of the command file, a Calibre extract deck, that defines the explicit devices.

**space_custom_args**

Use this field to enter extra command line flags for space.exe/space3d.exe. These should be inserted at the start of the args (immediately after the executable on the command line). Special arguments are documented in the PX_share_docs_folder. Note that HiPer PX does not perform error checking on these statements.
26 Viewing Verification Results

The Verification Error Navigator

The Verification Error Navigator provides an interface with which to browse and display DRC or Extract errors in the active layout. For both processes, the Verification Error Navigator is designed to simplify iterations of the error correction workflow.

The Verification Error Navigator is a resizable and dockable toolbar. It contains a scrollable tree of rules plus controls for loading errors and displaying them in the active layout. You can expand or collapse an individual rule or device to show or hide individual errors for that rule. For each DRC rule, L-Edit lists the number of violations in square brackets. Navigator controls are the same for both DRC and Extract results.

To open the Verification Error Navigator, select Tools > Verification Error Navigator > Show Verification Error Navigator from the L-Edit menu.

*Verification Error Navigator for DRC*


**Error Navigator Toolbar**

The following table lists the Verification Error Navigator toolbar commands. You can map shortcut keys for all commands using “Setup Application–Keyboard” on page 67.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Toggles the display of error markers on the selected error." /></td>
<td>Toggles the display of error markers on the selected error. When the display is on, L-Edit draws markers on the violating edge segments and shows an optional circle and cross hairs, as specified in the Verification Error Navigator Options dialog.</td>
</tr>
<tr>
<td><img src="image" alt="Removes an error marker completely, so that it can no longer be displayed." /></td>
<td>Removes an error marker completely, so that it can no longer be displayed.</td>
</tr>
<tr>
<td><img src="image" alt="Displays the next error in the tree. The shortcut for this command is the period key (.)" /></td>
<td>Displays the next error in the tree. The shortcut for this command is the period key (.) . Equivalent to Tools &gt; Verification Error Navigator &gt; Next Error.</td>
</tr>
<tr>
<td><img src="image" alt="Displays the previous error in the tree. The shortcut for this command is the comma key (,). Equivalent to Tools Verification Error Navigator &gt; Previous Error." /></td>
<td>Displays the previous error in the tree. The shortcut for this command is the comma key (,) . Equivalent to Tools Verification Error Navigator &gt; Previous Error.</td>
</tr>
<tr>
<td><img src="image" alt="Displays the Verification Error Navigator Options dialog to specify how errors are displayed. (See “Error Display Options” on page 950.) Equivalent to Tools &gt; Verification Error Navigator &gt; Verification Error Navigator Options." /></td>
<td>Displays the Verification Error Navigator Options dialog to specify how errors are displayed. (See “Error Display Options” on page 950.) Equivalent to Tools &gt; Verification Error Navigator &gt; Verification Error Navigator Options.</td>
</tr>
<tr>
<td><img src="image" alt="Toggles which rules are displayed in the navigation tree. When enabled, all rules are shown. When disabled, only rules with violations are shown. Equivalent to Tools &gt; Verification Error Navigator &gt; View All Rules." /></td>
<td>Toggles which rules are displayed in the navigation tree. When enabled, all rules are shown. When disabled, only rules with violations are shown. Equivalent to Tools &gt; Verification Error Navigator &gt; View All Rules.</td>
</tr>
<tr>
<td><img src="image" alt="Toggles which rules are displayed in the navigation tree. When selected, the top level of the tree is rules. When unselected, the top level of the tree is cells containing violations. Equivalent to Tools &gt; Verification Error Navigator &gt; View By Rule." /></td>
<td>Toggles which rules are displayed in the navigation tree. When selected, the top level of the tree is rules. When unselected, the top level of the tree is cells containing violations. Equivalent to Tools &gt; Verification Error Navigator &gt; View By Rule.</td>
</tr>
<tr>
<td><img src="image" alt="Displays each error in the cell in which it originally occurs. When this option is selected, errors are shown in the cell in which they occur. When unselected, all errors are shown in the top-level layout. Equivalent to Tools &gt; Verification Error Navigator &gt; Cell Context (see “Cell Context” on page 944.)" /></td>
<td>Displays each error in the cell in which it originally occurs. When this option is selected, errors are shown in the cell in which they occur. When unselected, all errors are shown in the top-level layout. Equivalent to Tools &gt; Verification Error Navigator &gt; Cell Context (see “Cell Context” on page 944.)</td>
</tr>
<tr>
<td><img src="image" alt="Displays a pull-down command menu of error navigator actions. See “Verification Navigator Command Menu” on page 944." /></td>
<td>Displays a pull-down command menu of error navigator actions. See “Verification Navigator Command Menu” on page 944.</td>
</tr>
<tr>
<td><img src="image" alt="Deletes the selected error from the tree and from the error layer, then selects and displays the next error in the tree. If a rule is selected, a dialog asks if you want to delete all errors for that rule." /></td>
<td>Deletes the selected error from the tree and from the error layer, then selects and displays the next error in the tree. If a rule is selected, a dialog asks if you want to delete all errors for that rule.</td>
</tr>
</tbody>
</table>

**Note:** Deletion of errors using ( ) cannot be undone.

Number of errors found in the current extract or DRC run.

You can view the number of marked errors (errors with checkmarks next to them) by moving the cursor over this field.
Chapter 26: Viewing Verification Results

The Verification Error Navigator

Verification Error Navigator for Extract

Legend:
- DRC or Extract Rule
- Cell name
- Error name
- Short or open circuit indicator
- Checkbox for "marking" errors
- [#] Number of violations corresponding to each rule or cell
- (#) Actual distance of each violation

Error Indicators During Verification Runs

L-Edit provides several visual indicators when a DRC or Extract job terminates prior to completion, usually due to an internal error, job cancellation or insufficient memory. When results in the Verification Error Navigator are incomplete, its background will be gold colored as shown below. A message is also written to both the summary report and the statistics report.
When a DRC or Extract job terminates unexpectedly as a result of running out of memory, L-Edit provide the following error message.

![Error message]

In addition, the “remaining” portion of the progress bar will be displayed in gold when there is a large amount of swapping during a verification run.

![Progress bar]

**Using Checkmarks**

Each rule, cell, and error shown in the Verification Error Navigator tree has a checkbox. You can mark items for any reason - for example, to indicate the errors which have been fixed, or to indicate errors that you want to review later. The Verification Error Navigator includes a number of preset actions that you can perform on the errors that are marked (see “Verification Navigator Command Menu,” below).

To add or remove a mark next to an item, simply click in the checkbox. When some but not all of the items in a certain group are checked, the checkbox for the parent item will have a grayed-out background.
Viewing Errors

For each error, L-Edit draws markers over the violating edge segments. Errors can be further highlighted with a circle and cross hairs, as illustrated below.

To display an error in the active layout, click on the Error name (·) in the Verification Error Navigator tree. L-Edit automatically pans and/or zooms to the error marker according to your settings in the View or Modify Verification Error Navigator Options dialog. (See “Error Display Options” on page 950.)

You can also use shift + click on a given error in the Verification Error Navigator to place a marker on that error in layout.

To step forward or backward through errors, click the Next Error (·) or Previous Error (·) toolbar buttons. (You can also select Next Error or Previous Error from the Tools > Verification Error Navigator submenu, or use the period (.) and comma (,) for shortcut key, respectively.

Click (·) to toggle marker display on or off for the selected error.

You can zoom in to enlarge and center the error in the layout window by pressing the shortcut key W. To return to the default view, click again on the Error name button.

Viewing “Job” Runs

The cell on which DRC or Extract is run defines a job. You can have multiple jobs loaded into the error navigator, but only one is displayed in the rule tree at a time. To switch between jobs (i.e., runs on different cells), use the drop-down combo box in the Verification Error Navigator toolbar (Row 1). A job is listed by the name of the cell on which it was run.

When you use the forward and backward buttons (·), you are scrolling through various combinations of job runs and active cell, as you would when using a web browser. When you create a new context by selecting a job or changing the active cell, the forward portion of the list of recent contexts is cleared.
**Cell Context**

Each error found in a job can be represented in two contexts. The cell on which the job was run defines the top-level context for that job. You can show an error as it appears in the top-level context for the current job (disabled), or you can show the error in the cell in which it occurs (enabled). The cell context is always the same for a given error, regardless of which job is active. If the error occurs in the top-level cell for the current job, then only one context is available.

**Verification Navigator Command Menu**

L-Edit provides a number of preset actions for marked errors, which you can access using the dropdown Command menu () on the Verification Navigator toolbar.

<table>
<thead>
<tr>
<th>Hide All Marked Errors</th>
<th>Show Hidden Errors</th>
<th>Clear All Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invert Marks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete Marked errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete Results...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Import Results...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Export Results...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Place Error Objects on Layout...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Place Device Labels on Layout...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open DRC Summary Report</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open DRC Runtime Statistics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Extract Summary Report</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Extract Runtime Statistics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>✔ Show DRC Results</td>
<td>✔ Hide All Marked Errors</td>
<td>✔ Show Hidden Errors</td>
</tr>
<tr>
<td>Show Extract Results</td>
<td>Clear All Marks</td>
<td></td>
</tr>
<tr>
<td>Show SQL Results</td>
<td>Delete Marked Errors</td>
<td></td>
</tr>
<tr>
<td>Delete Results</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hide All Marked Errors**

When this mode is selected, all marked errors are automatically hidden from the tree display. This mode is persistent — while you are in this mode, adding a check in the checkbox for an error causes it to be automatically hidden.

To exit the mode, select Hide all marked errors again to unmark it in the menu. This does not change the visibility or checkmarks of hidden errors; however, newly marked errors are not hidden when the Hide all marked errors option is off.

**Show Hidden Errors**

Shows errors that were previously hidden. This option does not expand the tree. Checkmarks are left unchanged.

**Clear All Marks**

Clears checkmarks from all errors in the tree. Hidden errors will remain hidden; use Show hidden errors to display these.

**Invert Marks**

Changes all error checkboxes to the opposite state — previously unmarked errors become marked, and previously marked errors become unmarked.

**Delete Marked Errors**

Permanently removes all marked errors from the DRC results. The Verification Error Navigator will issue a warning before deleting hidden errors.

**Delete Results**

Deletes all results for the current job.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Import Results</strong></td>
<td>Import DRC results from a text file. DRC Results from a command line HiPer Verify output, or from a Calibre® results database may be imported.</td>
</tr>
<tr>
<td><strong>Export Results</strong></td>
<td>Exports results for the current job to a text file. See “Exporting a Text File” on page 954.</td>
</tr>
<tr>
<td><strong>Place Error Objects on Layout</strong></td>
<td>Places objects (polygons, wires, and/or ports) on the error layer to highlight violations. See “Placing Error Markers,” below.</td>
</tr>
<tr>
<td><strong>Place Device Labels on Layout</strong></td>
<td>Places port names as device labels. See “Place ports as device labels” on page 917.</td>
</tr>
<tr>
<td><strong>Open DRC Summary Report</strong></td>
<td>Displays the DRC summary report in a text window. See “DRC Summary Report” on page 951.</td>
</tr>
<tr>
<td><strong>Open Extract Runtime Statistics</strong></td>
<td>Displays the Extract runtime statistics report in a text window. See “Extract Runtime Statistics Report” on page 958</td>
</tr>
<tr>
<td><strong>Show DRC Results</strong></td>
<td>Check this option to display DRC results in the Verification Error Navigator window.</td>
</tr>
<tr>
<td><strong>Show Extract Results</strong></td>
<td>Check this option to display Extract results in the Verification Error Navigator window.</td>
</tr>
</tbody>
</table>
**Verification Navigator Context Menu**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display in Cell Context</td>
<td>Shows errors in the cell where they originate.</td>
</tr>
<tr>
<td>Display in Top-level Context</td>
<td>Shows errors in their top-level cell context.</td>
</tr>
<tr>
<td>Mark</td>
<td>Enters a checkmark in each checkbox for the selected cell or rule and all those below it in the hierarchy.</td>
</tr>
<tr>
<td>Unmark</td>
<td>Clears the checkmarks in each checkbox for the selected cell or rule and all those below it in the hierarchy.</td>
</tr>
<tr>
<td>Jump to Rule in Setup DRC</td>
<td>In DRC, opens the design rule file at the highlighted cell or rule.</td>
</tr>
<tr>
<td>Jump to Device in Netlist</td>
<td>In Extract, opens the extract rule file at the highlighted device; see “Finding Devices and Nodes” on page 919.</td>
</tr>
<tr>
<td></td>
<td>From LVS or T-Spice you can also right-click on a line in the netlist and select <strong>Jump &gt; Device in Layout</strong>.</td>
</tr>
</tbody>
</table>

**Placing Error Markers**

When you select an error in the Navigator tree, L-Edit automatically highlights that error in the layout. (You can enter your preference for how a selected error is highlighted with “Error Display Options” on page 950.) These error markers, however, are temporary. When you select a new error, L-Edit clears the previous marker and highlights the currently selected error.

You can **shift+click** on a DRC error in the verification navigator to place an error marker on that error in layout.

If you wish to locate all errors in the layout, you can use the **Place error objects on layout** command to create error objects and/or ports for all violations. These error objects are added to the error layer and saved with the TDB file.
Error ports and objects can be moved, deleted, hidden, and shown in the same manner as other objects. Error objects are persistent; they are saved on the error layer of the TDB file and will remain there until you choose to delete them.

Placement options include:

- **Flat, all objects on top-level cell**: Places all error objects in the top-level layout for the DRC job.
- **In the cell associated with the error**: Places error objects in the cells in which they were found.
- **Examples, one instance of each error in the top-level cell**: Places a single error object for each rule that was violated. If an error is repeated in the layout, only one example will be shown with the error object. All example objects are placed in the top-level layout for the DRC job.

These additional options determine which objects are placed in the layout. You must select at least one of the first two options under **Objects to place** (polygons or ports).

- **Polygons and/or wires representing errors**: Places a polygon and/or wires on each error to illustrate the violation.
- **Ports labeling errors with text**: Places a port with text labeling each error. Each error port consists of the name of the violated design rule, the rule distance, and the spacing or nature of the error in brackets. For example, an error port named **8.4c Via to Active Spacing < 2 [1]** shows that the associated violation involves a spacing of 1, when the minimum Via-Active spacing should be 2.
- **Markers**: Places a circle, with crosshairs extending to the four sides of the window, on each error.

You can also use **shift + click** on a given error in the Verification Error Navigator to place a marker on that error in layout.
Setting the Color of DRC Markers

The error marker by default uses the last color in the color palette. The color palette can be accessed with Setup > Colors. You can change the last color by first selecting the color in the color list and then using the RGB slide bars or the color picker button to select an appropriate color for your error markers.

Finding Error Markers

You can use Edit > Find to locate marked errors. Specify a Wire search (for error objects) or a Port search (for error ports) on the Error layer. For ease of viewing, specify Pan or Pan & zoom as the viewing option.

Include errors hidden in the error navigator

Places error objects on all violations, including those that are hidden from display in the Verification Error Navigator tree. When this option is not checked, hidden errors are omitted from the error layer.

Note that because of the default rendering setup of the Error layer, a port’s text is only visible when the port is selected. Thus, as the search commands go through ports on the Error layer, the names of the error ports become visible one at a time.
Clearing Error Markers

Tools > Clear Error Layer opens the following dialog, prompts for options, then removes error markers (ports and objects).

![Delete Objects on Error Layer dialog]

Delete from:
- This cell (cell name)
- This file (file name)
- All files

WARNING: This operation cannot be undone.

This cell (cell name) Removes all objects on the Error layer in the active cell
This file (file name) Removes all objects on the Error layer in the active file
All files Removes all objects on the Error layer in all open files

Warning: This command cannot be undone.
Error Display Options

Viewing Options

Tools > Verification Error Navigator > Verification Error Navigator Options opens a dialog with controls for how L-Edit displays errors when they are selected in the Verification Error Navigator. You can also access this dialog by clicking ( ) on the toolbar.

The General tab contains these browsing and display options:

- **Display error marker circle**
  Instructs L-Edit to draw a circle around each error when it is being viewed.

- **Display error marker cross hairs**
  Instructs L-Edit to draw two lines - one vertical and one horizontal - that intersect the center point of the currently selected violation.

- **Reuse the active cell window**
  Displays the selected error in the active cell window.

- **Open the cell in a new window**
  Opens a new window when the selected error is not in the same cell as that shown in the active window (default).

- **Limit number of errors listed per rule-cell combination to:**
  Specifies the maximum number of errors included in the Verification Error Navigator tree for a single rule and cell.
### Sorting Options

The **Sort** tab contains options for sorting rules when they are viewed either by rule or by cell.

#### View or Modify Verification Navigator Options

![View or Modify Verification Navigator Options](image)

- **Descending error count**
  - Lists rules or cells in descending order of error count, with rules or cells containing the most violations listed first.

- **Alphabetically**
  - Lists rules or cells alphabetically.

- **Results order**
  - Lists rules in the order found in the DRC or Extract results (default). This option is active when **viewing by rule** only.

### DRC Report Files

L-Edit automatically generates both a summary report and a runtime statistics report for each DRC run. After DRC completes, these reports are stored with the cell on which DRC was run.

You can view these reports for the selected DRC job by selecting the corresponding menu options in the Verification Error Navigator Actions menu. You can also instruct L-Edit to display the summary report automatically upon completion of DRC, by selecting the corresponding option in either the **Design Rule Check** dialog or the **DRC Progress** dialog. When you display a report, L-Edit opens the report in an L-Edit text window with the default filename “Summary of *cellname.drc*” or “Statistics of *cellname.drc*.” You can then save the report to a text file, if desired.

### DRC Summary Report

The DRC summary report contains a summary of DRC results, including the number of errors found for each rule and each cell. The DRC summary report also gives information about memory usage, runtime, and files.

Following is a sample DRC Summary Report.

```
*************** RESULTS SUMMARY ****************************
DRC Errors Generated 215
CPU Time: 23:14:00
REAL Time: 00:01:00
Number of Input Objects: 5449 (70067)
Number of Rules Executed: 88
Number of Disabled Rules: 7
```
************* EXECUTION SUMMARY **********************
Execution Date/Time:       Jun 19 2002 10:50:49
L-Edit Version:           v10.0
Rule Set Name:            MOSIS/HP 1.0U SCN3M, Tight Metal
File Name:                
Cell Name:                top
User Name:                TannerEDA
Computer Name:            TANNERPC
Memory used at start:     OK

************* GEOMETRY FLAG SUMMARY *********************
ACUTE ANGLES............. 0
ALL ANGLE EDGES........... 0
OFFGRID ................... Disabled
SELF INTERSECTIONS ...... 1
WIRE JOIN STYLES .......... 0
WIRE END STYLES .......... 0

************* RULES WITH ERRORS FOUND *********************
1.3 Well to Well (Same Potential) Spacing ........................ 2
2.2 Active to Active Spacing .................................... 8
2.3b Source/Drain Active to Well Space ............................ 4
2.4a Well Contact (Active) to Well Edge .......................... 2
4.2a/2.5 Active to N-Select Edge ................................ 2
4.2b/2.5 Active to P-Select Edge ................................ 2
4.3c Not Exist: Active Contact not on act ........................ 4
...

************* RULES WITH NO ERRORS FOUND *********************
1.1 Well Minimum Width
1.2 Active Minimum Width
2.3a Source/Drain Active to Well Edge
2.4b Subs Contact (Active) to Well Spacing
...

************* DISABLED RULES *******************************
1.2 Well to Well (diff potential) Not che
2.5 Covered in 4.2. Active from diff imp
10.1b BondingArea:PadComment(100x100um)
...

************* CELLS WITH ERRORS FOUND ********************
PadVdd .......................................................... 2
Row_10 .......................................................... 2
TannerEDA Logo ................................................... 80
Chip ............................................................ 2
Frame ........................................................... 75
Pad_BidirHE08 ................................................... 1
PadGnd .......................................................... 52
Core ............................................................ 1

************* INPUT LAYER SUMMARY *************************
Active .............. Number of Geometry Objects = 155      (1577)
Active Contact ...... Number of Geometry Objects = 1222     (39830)
Cap Well ............. Number of Geometry Objects = 0        (0)
Metal1 .............. Number of Geometry Objects = 756      (3660)
...
The DRC summary report is organized into the following sections:

<table>
<thead>
<tr>
<th>RESULTS SUMMARY</th>
<th>The results summary includes the following information:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- DRC Errors Generated—the total number of errors generated,</td>
</tr>
<tr>
<td></td>
<td>including Geometry Flag violations.</td>
</tr>
<tr>
<td></td>
<td>- CPU Time and REAL Time—the cpu and wallclock times,</td>
</tr>
<tr>
<td></td>
<td>respectively. Each time is given in H:M:S format.</td>
</tr>
<tr>
<td></td>
<td>- Number of input objects—Number of objects obtained by</td>
</tr>
<tr>
<td></td>
<td>counting each primitive object in each cell. The number</td>
</tr>
<tr>
<td></td>
<td>of objects that would be counted in a flattened layout</td>
</tr>
<tr>
<td></td>
<td>is shown in parentheses.</td>
</tr>
<tr>
<td></td>
<td>- Number of Rules Executed—Total number of rules executed,</td>
</tr>
<tr>
<td></td>
<td>not counting geometry flags.</td>
</tr>
<tr>
<td></td>
<td>- Number of Disabled Rules—Total number of disabled rules,</td>
</tr>
<tr>
<td></td>
<td>not counting Geometry Flags.</td>
</tr>
</tbody>
</table>

| EXECUTION SUMMARY                | Lists statistics pertaining to the execution time, file   |
|                                  |   and cellnames, username, and memory usage.              |

| GEOMETRY FLAG SUMMARY            | The Geometry Flag Summary lists the number of Geometry     |
|                                  |   Flags found for acute angles, all-angle edges, off-grid |
|                                  |   vertices, self-intersecting polygons, and wire           |
|                                  |   join/end styles.                                       |
|                                  |   Acute angles, all-angle edges, and off-grid vertices     |
|                                  |   are only counted when their corresponding Geometry Flag   |
|                                  |   options have been checked in the DRC Setup dialog. If   |
|                                  |   these options are left unchecked, then they are listed  |
|                                  |   as “disabled” in the Geometry Flag Summary. Self-         |
|                                  |   intersecting polygons and wire join/end styles are      |
|                                  |   always reported.                                       |

| RULES WITH ERRORS FOUND          | Lists all rules for which at least one violation was       |
|                                  |   found. Rules are listed in the order they appear in the  |
|                                  |   Setup DRC dialog.                                       |

| RULES WITH NO ERRORS FOUND       | Enabled rules for which no errors were found.              |
|                                  |                                                            |

| DISABLED RULES                   | List of disabled rules.                                    |
|                                  |                                                            |

| CELLS WITH ERRORS FOUND          | List of cells in which errors were found, and the number   |
|                                  |   of errors in each cell (including geometry flag violations). |

| INPUT LAYER SUMMARY              | Lists each layer used by an enabled rule, including layers |
|                                  |   that do not contain any objects. Next to each layername,  |
|                                  |   the number of geometry objects is listed. The number of   |
|                                  |   objects on the layer if the database were flattened is     |
|                                  |   included in parentheses.                                 |

**DRC Runtime Statistics Report**

When you select Open DRC Runtime Statistics from Tools > Verification Error Navigator > Actions, L-Edit displays timing statistics for layer derivation and design rule checks in a text window. (You can see an example of a similar Extract statistics report in “Extract Runtime Statistics Report” on page 958.)
Exporting a Text File

You can export the results of a DRC or Extract job to a text file by selecting the option Export results in the Verification Error Navigator Command menu. L-Edit writes design rule errors to a text file with the cell name as filename and the extension .tde.

Displaying Calibre® DRC Results

L-Edit allows you to load DRC errors from a Calibre® DRC results database into the Verification Error Navigator and display them in the active layout.* The database you select may be flat or hierarchical, but it must be in ASCII format.

To specify the database to load, select Import Results from the Verification Error Navigator Command menu.

Type the Calibre DRC Results database filename in the editing field, or select Browse to navigate to the desired file.

You need to enter scaling information based on the technology you use in the Meters per User Unit field. For example, go to Setup > Setup Design > Technology to find your tech setup. If your tech units are one micron and the database resolution is 1/1000, the scaling factor will be 1e^-009 meters per user units.

Click OK to display the Verification Error Navigator for the specified file. The display shown in the error navigator depends upon the information in the Calibre file. If hierarchical information is included, the error navigator will display an error within the cell it originates and also in the cell on which the DRC was run.

Finding Open Circuits and Short Circuits

HiPer Verify warns in the Verification error navigator for open circuits and short circuits, with a display that lets you click on the warning in the navigator display to zoom directly to the error in layout. Warnings are grouped by type, cell and net name. for each error, the navigator shows the coordinate location, layer, and whether the conflicting object has been used or ignored.

For example, in the simple cell shown below, after extract the error navigator shows a short circuit, where a net has been labelled both GND and GND1, and an open circuit, where the same net name “A”

* Calibre is a registered trademark of Mentor Graphics Corporation.
is used three times. Each of these violations has a checkbox in which you can click to place an error marker to easily locate the violation in the layout.

### Extract Report Files

In addition to the netlist, L-Edit automatically generates both a summary report and a runtime statistics report for each Extract run. After Extract completes, these reports are stored with the cell on which Extract was run.

You can view these reports for the selected Extract job by selecting the corresponding menu options in the Verification Error Navigator Command menu. You can also instruct L-Edit to display the summary report automatically upon completion of Extract, by selecting the corresponding option in the **Extract Progress** dialog. When you display a report, L-Edit opens the report in an L-Edit text window with the default filename “Summary of *cellname*” or “Statistics of *cellname.Extract*.” You can then save the report to a text file, if desired.

### Extract Summary Report

The Extract summary report contains a summary of Extract results, including the number of errors found for each rule and each cell, and bad devices (ex. missing or extra pins). The Extract summary report also gives information about memory usage, runtime, and files.

Following is a sample Extract Summary Report.

```
L-Edit EXTRACT SUMMARY REPORT
EXECUTION SUMMARY Execution Start Time Feb 15 2008 11:04:05
L-Edit Version L-Edit Win32 13.00 Beta 3.20080215.07:18:40
Build Number 3749
Rule Set Name
File Name \suntan2\tmp\barry\HiPerExtractExample\lights.tdb
```
Cell Name: Pad_BidirHE  (Feb 15 11:03:56 2008)
User Name: jbergstr
Computer Name: DEVO1XP
Operating System: Windows XP 5.1 2600 Service Pack 2
Total physical memory: 2047.0MB
Memory used at start: 28.4M

EXTRACT JOB RESULTS SUMMARY
Total EXTRACT Errors Generated: 23
CPU Time: 00:00:00
Real Time: 00:00:08
Rules Executed: 21

EXTRACT Errors Generated by Rule Set
\suntan2\tmp\barry\HiPerExtractExample\HiPer-LVS.cal: 22
\suntan2\tmp\barry\HiPerExtractExample\Dracula025_4M.drc: 1

RUN-TIME ERRORS AND WARNINGS

L-Edit EXTRACT Log
PARSING SUMMARY
Command File: \suntan2\tmp\barry\HiPerExtractExample\HiPer-LVS.cal
Rule Set Name: Calibre LVS for Generic 0.25um process
No warnings.

Command File: \suntan2\tmp\barry\HiPerExtractExample\Dracula025_4M.drc
Rule Set Name: 
No warnings.

Running command file \suntan2\tmp\barry\HiPerExtractExample\HiPer-LVS.cal
Rule Set Name: Calibre LVS for Generic 0.25um process
Execution Start Time: Feb 15 2008 11:04:05
Maximum Results: 1000

INPUT LAYER SUMMARY
Layer Name: Object Count Flattened
Active: 7 7
ActiveCont: 410 410
Metal1: 30 30
Metal2: 10 10
NDiffResistorID: 1 1
NMOSCapacitorID: 0 0
NSelect: 3 3
NWell: 2 2
NWellResistorID: 0 0
PDiffResistorID: 0 0
PMOSCapacitorID: 0 0
PSelect: 4 4
PadComment: 1 1
Poly: 13 13
PolyCont: 17 17
PolyResistorID: 0 0
Via: 86 86

EXCLUDED CELLS
None

GEOMETRY FLAG SUMMARY
ACUTE ANGLES: Disabled
ALL ANGLE EDGES: Disabled
OFFGRID: Disabled
ZERO-WIDTH WIRES: 0
Chapter 26: Viewing Verification Results

Polygons with over 199 vertices 0
Wires with over 200 vertices 0
Self intersections 1
Wire join/end styles 0

------------------------------------------------------------------------
Cells with errors found Pad_BidirHE 22
------------------------------------------------------------------------
Results summary Errors generated 1
CPU Time 00:00:00
Real Time 00:00:02
Input objects 584 (584)
Rules executed 14
Geometry flags executed 5
Disabled rules 0

---------------------------------------------------------------
Running command file
\suntan2\tmp\barry\HiPerExtractExample\Dracula025_4M.drc Rule Set Name
Execution start time Feb 15 2008 11:04:08
Maximum results 1000

Input layer summary Layer Name Object Count Flattened

----------------------------------------------------------------------------
Excluded cells
None

----------------------------------------------------------------------------
Geometry flag summary Acute angles Disabled
All angle edges 0
Offgrid 0
Zero-width wires 0
Polygons with over 199 vertices 0
Wires with over 200 vertices 0
Self intersections 1
Wire join/end styles 0

----------------------------------------------------------------------------
Cells with errors found Pad_BidirHE 1

----------------------------------------------------------------------------
Results summary Errors generated 23
CPU Time 00:00:00
Real Time 00:00:06
Input objects 0 (0)
Rules executed 7
Geometry flags executed 7
Disabled rules 0

----------------------------------------------------------------------------

The extract summary report is organized into the following sections:

**Execution Summary** Lists statistics pertaining to the execution time, file and cellnames, username, and memory usage.
Chapter 26: Viewing Verification Results

Extract Report Files

Extract Runtime Statistics Report

When you select **Open Extract Runtime Statistics**, L-Edit displays run time and memory usage statistics for layer derivation and rules in a text window as shown in the following (truncated) example.

The Page Faults/Sec column indicates the amount of swapping during a job. If these values are high you should consider adding random access memory to your computer or using a 64-bit operating system.
Total physical memory:  2046.1MB  
Memory used at start:  22.2M

Command File: Y:\My Documents\doc\2008\torex.drc

************ RUNTIME STATISTICS ************

<table>
<thead>
<tr>
<th>Time</th>
<th>Cumulative Memory</th>
<th>Delta Memory</th>
<th>Faults/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT,</td>
<td>5.195s</td>
<td>30.0MB</td>
<td>+29400K</td>
</tr>
<tr>
<td>SIMPLIFY,</td>
<td>7.738s</td>
<td>41.8MB</td>
<td>+9936K</td>
</tr>
<tr>
<td>WS1W</td>
<td>0.512s</td>
<td>42.2MB</td>
<td>+388K</td>
</tr>
<tr>
<td>WS1W</td>
<td>0.007s</td>
<td>42.2MB</td>
<td></td>
</tr>
<tr>
<td>WS1S</td>
<td>0.022s</td>
<td>42.4MB</td>
<td>+184K</td>
</tr>
<tr>
<td>WS1S</td>
<td>0.002s</td>
<td>42.4MB</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>0.014s</td>
<td>42.5MB</td>
<td>+20K</td>
</tr>
<tr>
<td>,</td>
<td>0.001s</td>
<td>42.5MB</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>0.002s</td>
<td>42.5MB</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>0.001s</td>
<td>42.5MB</td>
<td></td>
</tr>
</tbody>
</table>

<etc.>

**Tanner “EVI” for Calibre Results Viewing Environment (RVE)**

Tanner Tools is integrated with the Mentor Graphics' Calibre Interactive and Calibre Results Viewing Environment (RVE) applications through the *External Verification Interface* (EVI) to provide a seamless mechanism for launching Calibre and viewing Calibre results in L-Edit or S-Edit.

EVI implements all of the Calibre RVE interface, so Calibre users can view DRC, LVS and parasitic extraction results, including highlighted nets and/or devices, in L-Edit and S-Edit. For more efficient LVS, EVI also takes advantage of Calibre RVE’s ability to display hierarchical SPICE netlists and to inspect parasitic capacitances, sorted by node or layer. You can also perform layout-to-schematic cross-probing from L-Edit and S-Edit, greatly simplifying LVS verification.

EVI resides entirely on the PC client running Tanner Tools and connects to a running Calibre RVE license on a Unix/Linux server. The single page EVI configuration dialog lets you set a port for communication with Calibre RVE and Calibre Interactive, the intermediate file locations for storing results when passing them from the PC to the server running Calibre, and the location of the L-Edit or S-Edit design database.

The External Verification Interface requires a separate subscription license, which can be purchased from Tanner EDA Sales.

**Installing and Configuring EVI**

You will need an X Window Server on your PC and a way to launch jobs on the remote Linux/Unix machine where Calibre® is located. Depending on your network and security setting, you can either set access controls option on the X server or Enable X11 Forwarding with your SSH client, but don’t set both.

See [Installing an X server](#) to set the Xming Access Controls Option for details on how to install a free X Window Server. [Installing a Telnet/SSH client](#) to Enable X11 Forwarding for details on how to install a free telnet/SSH client.
Starting Jobs on the Remote Machine

Follow these steps to test your connection and start a remote job.

[1] Start your X Windows Server

[2] Connect with Telnet or SSH to the Linux/Unix machine where Calibre is located.

[3] Once connected, type “export DISPLAY=*YourPCName*:0.0” at the prompt, where *YourPCName* is the name of your computer. (You can use the Windows key + the Pause (Pause) key to open the System Properties dialog, which has a Computer Name tab that shows your PC name.)

[4] Run XClock by typing “xclock” at the prompt. If XClock appears then you are able to connect and start a remote job.

Setup EVI

EVI runs on the Windows PC where you run L-Edit and S-Edit.

[1] Start EVI from the Windows computer where you run L-Edit and S-Edit. You can use either Start > All Programs > Tanner EDA > Tanner Tools v##.# > External Verification Interface or C:\ Program Files\Tanner EDA\Tanner Tools v##.#\evi.exe, where ##.# is the product version installed on your PC (e.g. 14.0). This will add the EVI icon to your system tray.

[2] If you get a Windows flag, allow the Firewall connection.
[3] In the EVI dialog, make sure the port number is 9189. You will need a location on the network for the results that both your Windows PC and Calibre can access.

![External Verification Interface](image)

- **Socket Port**: Enter 9189.
- **Calibre Path Prefix**: Type the path to the network location for the results in the form that Calibre (Linux/Unix) can use to access the directory (e.g. `/net/data/verification`).
- **Tanner Path Prefix**: Type the path to the network location for the results in the form that Tanner (PC) can use to access the directory (e.g. `\<servername>\<sharename>\verification`).
- **Start with Windows**: Check this box to launch EVI during Windows startup.

[4] Click **Apply** to accept your entries, then **Hide** to minimize the dialog to the system tray or **Exit** to close the dialog.

### Setup Calibre RVE

[1] Connect to the Linux/Unix machine where Calibre is located. (This is usually done with a Telnet or SSH client. If you don't have a telnet/SSH client, see “Installing a Telnet/SSH client to Enable X11 Forwarding” on page 964.)

[2] Start the Calibre toolbar by typing “`calibre - gui`” at the prompt. If this doesn't work, you may have to start the Calibre license manager first by typing “`lmgrd`”.

![Calibre Interactive](image)
[3] Click on RVE in the Calibre Interactive toolbar.

[4] Run Setup > Setup Layout Viewer in Calibre RVE.

[5] Set the Layout Viewer to Other and Tanner (L-Edit). (For older versions of Calibre RVE, you will need to set it to Other and Other.)

[6] Set the Hostname field to your hostname or the IP address of your Windows PC. The Socket Number should be 9189.

[7] Set the Highlight Data field to the same Calibre path prefix you set in the EVI dialog plus a filename to store the results. For example, /net/data/verification/query_results.

[8] Click the Connect button under IPC Settings to test the connection, then click OK.
[9] Run **Setup > Setup Schematic Viewer** in Calibre RVE.

![Setup Schematic Viewer](image)

[10] Set the **Schematic Viewer** to **Other** and **Tanner (S-Edit)**. (Again, for older versions of Calibre RVE you will need to set it to **Other** and **Other**.)

[11] Make sure the **Use layout viewer connection** option is enabled.

[12] Click **OK**.

**DRC Results Viewing**

[1] Unzip the test data to the network location indicated in the **Tanner Path Prefix** field of the EVI dialog.

[2] Start L-Edit and load the HiPer.tbd database file. L-Edit and S-Edit have to be running when viewing results in Calibre RVE.

[3] In Calibre RVE, load the DRC results file.

[4] Browse the DRC error database with Calibre RVE.

**Installing an X server to set the Xming Access Controls Option**

[1] Install Xming X Server on your PC (http://www.straightrunning.com/XmingNotes/).


[5] To set the access controls on the X server, right-click on the Xming desktop icon and add `-ac` after `:0` in the Target field.

![Xming Properties](image)


**Installing a Telnet/SSH client to Enable X11 Forwarding**

[1] Install PuTTY from (http://www.putty.org/).

[2] Start PuTTY.

[3] To enable X11 forwarding select **Connection > SSH > X11** category.

[4] Check the **Enable X11 Forwarding** checkbox only if you didn’t use the access control option on your X windows server.

[5] Leave the **X display location** blank. PuTTY will automatically fill this in.

![PuTTY Configuration](image)
**EVI for DRC and LVS**

You can run DRC or LVS directly on a cell in L-Edit or S-Edit by specifying the design database and the primary cell in Calibre Interactive. Communicating through the EVI plug-in, Calibre Interactive then exports the files needed from L-Edit or S-Edit and runs Calibre on those files.

- For DRC, EVI makes a call to L-Edit to export the layout in GDSII format, then sends it to Calibre nmDRC.
- For LVS, EVI makes a call to L-Edit for the exported layout and to S-Edit to export the schematic in SPICE format, then sends them to Calibre LVS. Calibre runs and then creates the Calibre results file, while the user remains in L-Edit. When the results are ready, they are loaded into Calibre RVE for review.

**Viewing DRC Errors**

To view DRC errors you click on the error in Calibre RVE, which communicates through EVI to highlight the error in L-Edit. The errors display in sequence, and you resolve each one in turn in L-Edit, optionally re-running DRC.

**Cross Probing**

EVI supports both tools during LVS by requesting a given net in the layout or schematic and highlighting it in L-Edit and S-Edit. EVI can also query RVE for the location of particular devices in the layout or schematic by device name or from the cursor’s current position.

**Shorts and Open Circuits**

To track down shorts and opens on complicated nets in layout, EVI can highlight a net by marking only specified layers or by marking each layer of the net in a different color.

**Finding Parasitics**

You can also browse extracted parasitic results in Calibre RVE and L-Edit or S-Edit, sorting parasitic results by net name or by value to find those nets of greatest interest. EVI can highlight the resistance, capacitance to substrate, and coupling capacitance in the layout or schematic. This allows you to pinpoint the specific layout geometry causing the largest parasitics, then modify the layout to minimize any parasitic effects.
27 Cross Probing

Tanner Tools provides easy and efficient verification with cross probing that lets you click in a netlist to locate corresponding nets and devices in the layout or schematic.

When you select a device from an extracted or exported netlist, you can instantly view that object in a layout view in L-Edit, or a schematic view in S-Edit. Because Tanner applications produce hierarchical netlists for designs with various levels of hierarchy, you can locate a problem quickly and resolve it just once, in the cell where the error originates.

**Key Features**

- You can cross probe to layout or schematic from any of the Tanner applications that reads a SPICE netlist—LVS, S-Edit, T-Spice or L-Edit—by right-clicking on a device in the netlist.
- From a SPICE file open in any Tanner text editor, you can bookmark all instances of all instances of a subcircuit, go to the start or end of a subcircuit, jump to the definition of a subcircuit from an instance of a subcircuit, or jump to the model statement from an instance of a primitive device.
- You can set the level of zoom and type of highlighting you prefer, and use standard L-Edit marker controls to control your cross probing markers.

**Using Cross Probing**

To cross probe to layout, use a netlist extracted from L-Edit. To cross probe to a schematic, use a netlist exported from S-Edit. To cross probe to either schematic or layout use an LVS results file. The design you are cross probing must be active in L-Edit for layout or S-Edit for schematic.

You will notice that extracted netlists include a comment at the end of each device, which is a set of coordinate values for the displacement, rotation and array values used to locate the device in layout. Schematics use the instance name to locate instances.

Remember that the design must be open and active in the intended application for cross probing to work. When you select a cross probe option, the corresponding L-Edit or S-Edit taskbar button will blink to alert you that the result is available.
Right-click on a device in the netlist to open the menu shown below and click on Jump. (Only Tanner options are described below, all others are standard Windows operations.)

![Menu](image)

1. **Insert Single Line Comment ("...")**
   - When the cursor is midline, inserts the character for a mid-line comment, a dollar sign ($), at the cursor position. Disabled when the cursor is at the beginning of a line.

2. **Remove Single Line Comment ("...")**
   - When the cursor is on a comment line, removes the start of line comment character, an asterisk (*), from that line.

3. **Jump**
   - Opens the crossprobe menu (shown below).

4. **Refresh Syntax Highlighting**
   - Re-applies netlist formatting.

Click on Jump to open this menu and select the option you want.

- **Device in Layout**
  - Opens the cell in L-Edit and draws a marker on the device.

- **Device in Schematic**
  - Opens the cell in S-Edit and highlights the marker.

- **Net in Schematic**
  - Highlights the node in S-Edit.

Alternately, you can highlight a device in L-Edit then use Tools > Jump to “devicename” in schematic to highlight the corresponding device in S-Edit.
[4] S-Edit will display the device in the selected state.

If the proper design is not open in L-Edit, or if you have attempted to probe from an exported netlist, you will get an error in the window. For example,

Probe: Device "MP1" not found in cell "NOR2"

If the proper design is not open in S-Edit, or if you try to probe from an extracted netlist into schematic, S-Edit will display this error message.
[6] In L-Edit, the device will be highlighted in its most primitive cell in the design hierarchy. To help locate the cross probed device, you can press \textbf{W} to zoom into it, or you can use the \textbf{Toggle Marker} button to turn the highlighting on and off.

Marker colors in L-Edit are determined by the settings for node highlighting markers (see “Node Highlighting Setup” on page 517).

[7] When you pick view schematic options from a netlist file, S-Edit will open the corresponding page with the selected device highlighted. For example, S-Edit will highlight instance P2 of cell PMOS in NAND2C when you jump to \textbf{Device in Schematic} from the following line:
MP1 Out1 A Vdd Vdd PMOS W=2.5u L=250n M=2 AS=4.5p PS=13.6u AD=3.125p PD=7.5u
LVS, an abbreviation of *layout versus schematic*, is a netlist comparison tool. It compares two netlists to determine whether they describe the same circuit. When they do not, LVS works in conjunction with L-Edit to identify and correct errors.

LVS can be used to determine whether a schematic circuit matches a layout, or whether two different schematics or layouts implement the same circuit.

**LVS Features**

- **SPICE input format** LVS accepts standard SPICE-format netlists.
- **Fragmentation identification** When two netlists are not equivalent, LVS can identify unresolvable nodes and devices and assist in locating them on the original schematic or layout.
- **Automorphism resolution** LVS identifies *automorph classes*—sets of elements or nodes (such as devices in parallel) which cannot be distinguished from one another. To resolve automorph classes, LVS can either employ user-supplied *prematch* information or a detailed *trial matching* process.
- **Parameter comparison** LVS uses *topological* (device types, number of connections), *parametric* (resistance, capacitance), and *geometric* (area, length, width) information to compare netlists. Matching thresholds can be defined to specify how different two values can be while still comparing equivalently. Different margins can be defined for parametric and geometric comparisons.
- **Permuted class resolution** LVS can identify the switching of two elements in series.

**Launching LVS**

You can launch LVS by:

- Clicking the **Start** button on the Windows toolbar and navigating to the application through the **Programs** menu
- Double-clicking the LVS icon from your desktop or another Tanner tool
- Double-clicking an LVS verification database (.vdb) file in Windows Explorer
- Double-clicking an LVS batch (.bat) file in Windows Explorer (see “Using LVS in Batch Mode” on page 1004)
- Invoking LVS from a DOS command prompt (see “LVS Command-Line Syntax” on page 1030.)
The LVS icon looks like this:

![LVS Icon]

## Input and Output Files

LVS requires two SPICE-format netlist files for input. SPICE files can be in either T-Spice or P-Spice format. In T-Spice mode, LVS also accepts netlists in H-Spice or Berkeley Spice format.

Optional input files include prematch files and element description files.

Information required for the verification run—input and output files, plus verification options—is referred to as a verification setup. LVS saves a setup to a verification database (.vdb) file. Multiple verification setups can be exported to a batch (.bat) file for subsequent invocation from a DOS command prompt. See “Setup—Input” on page 977 and “Setup—Output” on page 978.

### Prematch Files

A prematch file is a text file that specifies pairs of nodes and elements that are to be considered equal. LVS uses this information to resolve automorph classes.

**Syntax**

Each line of a prematch file names two nodes or elements that are to be considered equal:

```
member1a member2a
member1b member2b
member1c member3c
...
```

- **member1** … The entry on the left in each line belongs to the **Layout netlist** specified in the Run LVS dialog—see “Setup—Input” on page 977.
- **member2** … The entry on the right in each line belongs to the **Schematic netlist** specified in the Run LVS dialog—see “Setup—Input” on page 977.

The prematch file may also contain a single node or element name on a line, in which case the same name will be used for matching layout and schematic.

### Element Description Files

The element description file .elm contains a list of statements defining all non-SPICE devices present in the netlists to be compared by LVS. It is only needed when the netlist files being compared contain special devices.

**Syntax**

Element description statements have the following syntax:
|| d name pin [pin ...] {{perm[,perm ...]}};

An element description statement always begins with these characters.

**name**

The name of the element being defined. The element name must match the subcircuit name used in the SPICE file.

**pin**

A list of the pins for the element. There must be at least one pin.

**perm**

An optional list of pin permutability statements, enclosed in parentheses. If more than one permutability statement is included, then they are separated with commas. No spaces are allowed within the parentheses.

; An element description statement always ends with a semicolon.

**Permutability Statements**

Pin permutability statements are used to indicate pins whose order can be swapped. For example, if you were defining a transistor, it might not be possible to distinguish between the source and drain pins, so you would declare them permutable. The names of the pins used in these statements must *exactly* match the pin names used elsewhere in the statement, including capitalization. Pin permutability statements can have one of two forms:

*pin*=

or

\[pin1=pin2=[pin3=[...]]\]

The first example states that all pins with the given name (*pin*) are permutable. The second statement states that all pins in the list (*pin1, pin2, pin3, …*) are permutable with each other. No spaces are allowed in either form.

**Element Description Examples**

|| d C POS NEG (POS=NEG) ;

The example above defines a non-polarized capacitor.

|| d DEV G1 G2 G3 DS DS (G1=G2=G3,DS==) ;

The example above defines a custom element, which will be called in the netlist file through a subcircuit element statement.

**File Locking**

LVS reads in all input files, including layout and schematic netlist files, element description files, and prematch files, at the start of a verification run. You can open such files for reading at this time, but LVS will prevent you from editing them. The verification database (*vdb*) file itself is locked throughout the verification, and you cannot edit it at any time during the run.

LVS also locks input files while processing each verification setup in a queue or batch file.
Backup Files

LVS automatically creates a backup of each verification database file, using the same filename as the original, but with a `.vdo` extension. Each time you modify the `.vdb` file, LVS will overwrite the previous backup of that file.

User Interface

The basic LVS user interface (shown below) consists of the following elements:

- Title bar
- Menu bar
- Toolbar (optional)
- Status bar (optional)

Depending on the types of files that are currently open and the task being performed, the user interface may also contain the following other elements:

- “Setup Window” on page 976
- “Text Window” on page 989
- “Verification Window” on page 990
- “Verification Queue” on page 992
Menus

The availability of individual menus depends on the types of files you have open in LVS. When no files are open, the LVS menu bar will look like this:

Commands for creating, opening, saving, and printing files

Commands for displaying or hiding elements of the user interface

Commands for starting, stopping, or setting up verification runs

Commands for starting, stopping, or setting up verification runs

Commands for accessing the online documentation

When the active window contains a setup file, the LVS menu bar will look like this:

Commands for selecting, tiling, or cascading windows

When the active window contains a text file, the LVS menu bar will look like this:

Commands for editing and searching in text files

Toolbar

You can position the toolbar anywhere within the application window or dock it against one side of the application window. You can also display or hide the toolbar by selecting View > Toolbar.
Chapter 28: Introduction to LVS User Interface

The availability of toolbar buttons will depend on the type of file in the active window—setup or text—and whether or not a verification queue exists.

Status Bar

When a verification window is active and the cursor is in the output file section, the status bar displays the number of the line where the cursor is positioned. If a text window is active, the status bar displays both the line number and column number of the cursor position.

Setup Window

When one or more LVS setup files are open, the user interface will contain one setup window for every open setup file. The setup window contains five tabs, and it is used to specify input files and a variety of verification options.

To create a new setup window, select File > New. In the New dialog, select LVS Setup and click OK.

LVS will create a new setup window with the default name Setup1, which corresponds to the setup file Setup1.vdb. When you save the setup file for the first time, you can either use the default name or supply your own, but you must use the .vdb extension (to ensure that Windows automatically recognizes the files as an LVS verification database). Subsequent new setup windows will receive similar names by default—Setup2, Setup3, etc.

To open an existing setup window, select File > Open. In the Open dialog, select the appropriate Verification Database File (*.vdb) from the Files of type drop-down list and select an available file or enter its name in the File name field.
Setup—Input

The **Input** tab contains fields for specifying input files to be compared by LVS. You can type the correct filename in a field, or use a **Browse** button to navigate to an appropriate directory. Click **Edit** to open the specified file in a text window.

**Layout netlist** and **Schematic netlist**

The two input files to be compared. Each file must be in a SPICE format chosen from the drop-down list provided. Options are:

- **T-Spice** — Select T-Spice syntax. This is the default mode.
- **H-Spice** — Select HSPICE syntax.
- **P-Spice** — Select P-Spice syntax. This mode must be selected if the input files were produced by P-Spice.
- **CDL** — Select Cadence Circuit Description Language syntax.

The default filename extensions is `.spc` for both input netlists. (See “SPICE File Format” on page 994.)

**Prematch file**

An optional input file that specifies equivalent elements and nodes for the iterative matching process. The default filename extension is `.pre`. (See “Prematch Files” on page 972.)

**Element description file**

An optional input file containing instructions on how to deal with custom devices. The default filename extension is `.elm`. (See “Element Description Files” on page 972.)

**Save modified files without prompting**

If this option is enabled, text files that are open and modified within LVS and are required for the LVS job will be automatically saved when “Run LVS” is pressed.
**Setup—Output**

The **Output** tab contains fields for specifying output files and display options.

In the **Output Files** section, you can type the correct filename in a field, or use a **Browse** button to navigate to an appropriate directory. Click **Edit** to open the specified file in a text window.

- **Output file**
  - When checked, specifies an output file containing verification results.
  - The default filename extension is `.out`.

- **Node and element list**
  - An optional output file containing a node and element list. The default filename extension is `.lst`. 

Chapter 28: Introduction to LVS

Flattened layout netlist and Flattened schematic netlist

Optional output files containing the flattened layout and schematic netlists. Each file includes only the information that LVS will use to compare circuit descriptions. Commands, comments, and ignored devices or parameters are omitted, and parallel or series elements are merged as specified in the top section of the Merge Devices tab.

Note: If Merge series MOSFETs is selected on the Merge Devices tab, the flattened netlist will not reflect collapsed series MOSFETs. To maintain SPICE compatibility, LVS writes the flattened netlist before merging series MOSFETs.

The default filename extensions are .spc and .sp for the flattened layout and schematic netlists, respectively.

Flattened netlists use the @ symbol instead of parentheses ( ) to designate hierarchy in device and net names. For example, M1@X1/X2/ for devices and N1@X1/X2/ for nets instead of M1(X1/X2/) and N1(X1/X2/). This syntax is compatible with T-Spice and H-Spice.

Overwrite existing output files

When checked, causes automatic overwriting of the output files, even if these files already exist.

Display Options

Use the checkboxes to specify which processing information LVS will display and/or record during the verification run:

- **Output file**—when checked, prints the selected information to the output file. To enable this option, you must specify an **Output file** in the top portion of the dialog.

- **Screen**—when checked, displays the selected information in the verification window.

**Show fragmented classes**—Shows information on fragmented classes of nodes and elements.

**Show automorph classes**—Shows information on automorph classes of nodes and elements.

**Show series MOSFETs that differ in order**—Shows information on functionally equivalent groups of series MOSFETS that differ in order (also known as permuted classes).
Setup—Device Parameters

The Device Parameters tab allows you to specify which parametric information LVS should consider for different elements during iteration matching and detailed trial matching.

![Diod.db](image)

**Show detailed processing information**—Instructs LVS to include additional processing information:

- Shows single-connection nodes.
- If a prematch file is used, displays prematched elements and elements that LVS attempts to postmatch. This is useful for troubleshooting netlists returned as not identical due to fragmentation after automorphism or permutability.
- If an element description file is used, lists subcircuits that will not be flattened (i.e., those designated as special elements).
- Logs merged series or parallel devices.
- Lists deletions of shorted and disconnected devices.
- Lists parasitic devices that were removed or shorted.

For further information on individual netlist classes, see also “Resolving Fragmented Classes” on page 1024, “Resolving Automorph Classes” on page 1025 and “Permutated Classes in Digital Designs” on page 1026.

**R, C and L Elements:**

<table>
<thead>
<tr>
<th>Device parameters to match:</th>
</tr>
</thead>
<tbody>
<tr>
<td>R, C and L Elements</td>
</tr>
<tr>
<td>MOSFET Elements</td>
</tr>
<tr>
<td>B, D, J, Q and Z Elements</td>
</tr>
<tr>
<td>T Elements</td>
</tr>
<tr>
<td>X Elements (Unflattened subcircuits)</td>
</tr>
<tr>
<td>Deviations (in %)</td>
</tr>
</tbody>
</table>

- Maximum element-value tolerance: 5.000
- Maximum geometric-value tolerance: 5.000
### MOSFET Elements:
- **Lengths and widths**
  Considers MOSFET length and width parameters.
- **Area and perimeter of source and drain**
  Considers source/drain areas and perimeters of MOSFET elements.
- **Diffusion squares of gate, bulk, source, and drain**
  Considers parametric information about diffusion squares of MOSFET gate, bulk, source, and drain terminals.

### B, D, J and Q Elements:
- **Areas**
  Considers area parameters for non-MOSFET semiconductor devices.

### T Elements:
- **Transmission line impedance**
  Considers transmission line impedance.
- **Transmission delay, line frequency and number of wavelengths**
  Considers transmission line parameters describing delay, frequency, and wavelength number.

### X Elements:
- **Parameters**
  Considers parameter values of unflattened or autodefined subcircuits. These parameters are treated as “value” parameters for the purpose of comparison tolerance. Parameters of “regular” subcircuits are passed down the hierarchy to the contents of each subcircuit.

### Deviations (in %):
- **Maximum element-value tolerance**
  The maximum amount (as a percentage of the larger value) by which two parameter values may differ and still compare as equal. The default is 5%.

  This tolerance applies to:
  - Resistance, capacitance, and inductance values
  - Transmission line number of wavelengths, frequency, delay, and impedance
  - MOSFET numbers of gate/source/bulk/drain diffusion squares

- **Maximum geometric-value tolerance**
  The maximum amount (as a percentage of the larger value) by which two geometric shapes may differ and still compare as equal. The default is 5%.

  This tolerance applies to:
  - GaAsFET, diode, JFET, and BJT element areas
  - MOSFET length and width
  - MOSFET source/drain areas and perimeters

For more information, see “Parameter Matching” on page 1026.
Setup—Merge Devices

Before LVS compares the two input netlists, it can eliminate potential ambiguities by reducing the total number of devices used in each circuit. LVS does this by merging similar devices in series or parallel configurations. (In previous versions of LVS, merging devices was referred to as network optimization.) When LVS merges a group of series or parallel devices, it replaces them with a single equivalent component.

Merge options for each device type are specified in the Merge Devices dialog.

Devices That Can Be Merged

The top portion of the dialog controls merge options for the following device types and configurations:

- Resistors (R) — parallel and series
- Inductors (L) — parallel and series
- Capacitors (C) — parallel and series
- GaAsFETs (B) — parallel
- Diodes (D) — parallel
- JFETs (J) — parallel
- BJTs (Q) — parallel
- MESFETs (Z) — parallel
- MOSFETs (Mp, Ms) — parallel MOSFETs and stacked MOSFETs (series MOSFETS with gates connected to the same node). For parallel MOSFETS to be merged, both devices must have the same gate length; the gate width of the resulting device is the sum of the widths of the original devices, and the area and perimeter of the source and drain are the sum of the respective terminal dimensions. For stacked MOSFETS to be merged, both devices must have the same gate width; the
gate length of the resulting device is the sum of the lengths of the original devices. In the case of merged stacked MOSFETS, the area and perimeter of the diffusion between the devices is divided evenly between the resulting source and drain terminals.

For each device type, select one of the following options from the drop-down list:

**None**
Leaves devices in their current configuration. LVS will not merge devices for which None is selected.

**All**
Merges series or parallel instances of the same device model into equivalent single devices.

**Model**
When the Model option is selected, LVS merges only devices that are instances of the specified model(s). In the text entry field, type the models you wish to make available for merge operations. List models with the following syntax:

```
type_name1, type_name2, ...
```

where type is the letter abbreviation for the device type, and name1, name2, etc. are model names defined in .model statements.

Note that merges of resistors can result in no devices, or with one device with disconnected terminal(s).

This is because The LVS merge algorithm works by iteratively collapsing parallel and series devices. Sometimes, the order in which these merges take place lead to unexpected results. For example, consider the simple case of two resistors in parallel. If the parallel merge takes place first, we are left with one resistor (and two nodes, one at each "end" of the resistor). However, if the series merge takes place first, we are left with one resistor, with it’s two ends shorted together.

Furthermore, the values of the remaining resistors are quite different in these two cases. This situation can become further confused, depending on the settings of the Remove shorted devices and Remove disconnected devices options. Either of these two cases can then wind up with zero devices (and zero nodes).

Fortunately, in typical circuits, these ambiguities are resolved by other circuit elements that are connected to the resistor array. If, in the previous example, the two nodes of the parallel resistor pair are each connected to, say, a capacitor, the series optimization is no longer possible, leaving only the parallel merge option.

**Merging Series MOSFETS**

You can use the Set All or Set None button to quickly change all merge settings except for the series MOSFETs.

**Series MOSFET optimization (treat series MOSFETS as logically equivalent)**
Specifies merge behavior for MOSFET devices in series. If this option is selected, the order in which series MOSFETs occur is ignored.
Select None, All, or Model in the pull-down menu. If Model is selected, type the names of the models to be considered for merge operations in the editing field.
Find series MOSFETs that differ in order or parameter values

When **Series MOSFET optimization** is set to either **All** or **Model**, this option instructs LVS to identify functionally equivalent groups of series MOSFETs that occur in different orders. The default state for this option is unchecked.

When this analysis is enabled, the parameter values of the series MOSFETs are inspected after the matching is complete. Parameter mismatch warnings are given in cases where the differences exceed the comparison tolerance specified in the Device Parameters tab.

**Note:** Series MOSFETs that differ in order are also called **permuted classes**. See “Permuted Classes in Digital Designs” on page 1026 for further discussion.

Remove shorted devices

Deletes shorted devices, in which all terminals are connected together.

Remove disconnected devices

Deletes **disconnected** (open) devices. A MOSFET is considered disconnected if the gate terminal is not connected to any other device, and at least one source/drain terminal is not connected to any other device. All other devices types are considered disconnected if at least one of their terminals is not connected to any other device.

**Setup Window—Parasitics**

The **Parasitics** tab contains fields to help LVS identify parasitic capacitors and resistors that should not be considered in the netlist comparison.

For example, parasitics may have been added to one of the input netlists for detailed timing simulations. These elements must be removed prior to netlist comparison. Similarly, even with a **.include** statement, LVS will not consider nodes to be the same if they are named differently but not connected.

LVS automatically shorts out the following nodes and removes the voltage source or resistor prior to netlist comparison:

- nodes connected by a 0V voltage source
- nodes connected by a resistor of model **R_SHORTED_VSOURCE**
- nodes named in a **.connect** statement.

When you use the **.CONNECT** command to short two nodes the syntax can be in the form of either **.connect C1@X1/ C2@X1/, .connect X1/C1 X1/C2**, **.connect C1(X1/) C2(X1/)**.

(Prior to version 14.00, LVS would look for voltage sources with the syntax **Vsourcename node1 node2 0** and convert them to zero-ohm resistors having the model name **R_SHORTED_VSOURCE**. These could then be filtered out using a very small value (ex. 1.0000e-006) in the “**Short out resistors**” field, or by entering **“R_SHORTED_VSOURCE”** in the “**Short out device models named**” field. When LVS added support for voltage source syntax with the DC term and voltage units in the voltage source device definition (**“R1 a b dc 0”** as well as **“R1 a b 0V,”** where the “0” can include a SPICE suffix.) LVS would convert the 0V voltage source to a 0ohm resistor, and you would short out a zero-volt voltage source by shorting all of its terminals and then removing the device.)
When specifying models, use the syntax \texttt{type\_namex}, where “type” is the letter abbreviation for the device type, and “name1,” “name2,” etc. are model names defined in the \texttt{.model} statements.

\begin{itemize}
\item \textbf{Short out resistors} \(\leq\) Removes any resistor with resistance less than or equal to the specified value, and connects (shorts) the two nodes that were spanned by the device.
\item \textbf{Remove resistors} \(\geq\) Removes any resistor with resistance greater than or equal to the specified value.
\item \textbf{Remove capacitors} \(\leq\) Removes any capacitor with capacitance less than or equal to the specified value.
\item \textbf{Short out capacitors} \(\geq\) Removes any capacitor with capacitance greater than or equal to the specified value, and connects (shorts) the two nodes that were spanned by the device.
\item \textbf{Remove device models named:} Removes all instances of the device models listed in the text entry field. Use the syntax \texttt{type\_namex}, where “type” is the letter abbreviation for the device type, and “namex” is a model name.
\item \textbf{Short device models named:} Removes any instance of the device models listed, and connects the nodes that were spanned by the device. Use the syntax \texttt{type\_namex}, where “type” is the letter abbreviation for the device type, and “namex” is a model name.
\end{itemize}
Setup—Options

The **Options** tab contains options for parsing files and flattening netlists, as well as an option to find soft connections with specified models (see “Detecting Soft Connections with LVS” on page 987): 

- **Consider M bulk terminals and B, J, Q, Z substrate terminals**
  - When checked, considers the fourth terminal on MOSFET, BJT, GAASFET and MESFET devices during verification. If unchecked, this terminal is ignored. The default is checked, to consider bulk nodes.

- **FET S and D are swappable, except:**
  - When checked, the source and drain terminals of FETs (MOSFETs, GAASFETs and MESFETs) are considered to be interchangeable, except for device models that are explicitly listed. When not checked, the reverse is true: by default, FETs are asymmetric (e.g., LDD devices), and symmetric device models are listed explicitly. The default state is checked.

- **BJT C and E are swappable, except:**
  - When checked, the collector and emitter terminals of BJTs are considered to be interchangeable, except for device models that are explicitly listed. When not checked, the reverse is true: by default, BJTs are asymmetric, and symmetric device models are listed explicitly. The default state is unchecked.

- **Resistors are polarized, except:**
  - When checked, all resistors are regarded as polarized elements (i.e., the two terminals are not interchangeable), except for device models that are explicitly listed. When not checked, the reverse is true: by default resistors are unpolarized, and polarized models are listed explicitly. The default state is unchecked.
Detecting Soft Connections with LVS

A MOSFET is said to be “soft connected” to a node when the only electrical connection to that node goes through a well or a substrate. This condition can be a serious problem if, for example, instead of being connected directly to a power supply metallization, a transistor’s source is connected only to the well (through an ohmic contact) and that well is in turn connected to the power rail (through another ohmic contact). Soft connections can lead to reduced drive strength, longer gate propagation delays, and susceptibility to latch-up.

LVS has an option to detect such soft connections when special devices are included in the SPICE netlist. It is recommended that ohmic contacts to wells and substrate be extracted as zero-ohm resistors, of type `R_WELLCONTACT` and `R_SUBSCONTACT`. (See “Detecting Soft Connections with LVS,” below. Enter the model names of the devices used to identify ohmic well and substrate contacts.

To detect soft-connected devices, check the option labeled **Find soft connections with models:** and enter the model names of the devices used to identify ohmic well and substrate contacts.

Before executing the regular LVS netlist comparison algorithm, LVS searches for wells and substrate that connect to at least one device (excluding MOSFET bulk terminals) and connect through soft-contact devices to more than one node. LVS prints the names of all nodes that are soft-connected through each well/substrate. LVS then shorts and removes all soft-connect devices, and proceeds with the regular netlist comparison algorithm.

<table>
<thead>
<tr>
<th>Capacitors are polarized, except:</th>
</tr>
</thead>
<tbody>
<tr>
<td>When checked, all capacitors are regarded as polarized elements (i.e., the two terminals are not interchangeable), except for device models that are explicitly listed. When not checked, the reverse is true: by default capacitors are unpolarized, and polarized models are listed explicitly. The default state is unchecked.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inductors are polarized, except:</th>
</tr>
</thead>
<tbody>
<tr>
<td>When checked, all inductors are regarded as polarized elements (i.e., the two terminals are not interchangeable), except for device models that are explicitly listed. When not checked, the reverse is true: by default inductors are unpolarized, and polarized models are listed explicitly. The default state is unchecked.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Find soft connections with models:</th>
</tr>
</thead>
<tbody>
<tr>
<td>When checked, LVS performs a pre-search for wells and substrate contacts that connect as described in “Detecting Soft Connections with LVS,” below. Enter the model names of the devices used to identify ohmic well and substrate contacts.</td>
</tr>
</tbody>
</table>
Setup—Performance

The **Performance** tab contains options that instruct LVS how to perform during iteration.

- **Normal iteration: consider fanout and element type**
  - Considers both fanout and element type during iteration. This is the default.

- **Fast iteration: consider fanout only**
  - Considers only fanout.

- **Continue processing when mismatch in node or element count found**
  - Continues to run even on element or node count mismatch. If this option is not selected, LVS issues a prompt when an element/node count mismatch occurs. In batch or queuing mode, this option is automatically selected.

- **Automatically perform detailed trial matching to resolve automorph classes**
  - Instructs LVS to perform detailed trial-matching when automorphed element or node classes occur without first issuing a prompt. In batch or queuing mode, this option is automatically selected.
Text Window

When a SPICE file, or other text-format file, is open, the user interface will contain one or more text windows.

The text window is used to edit text-format files, such as SPICE, prematch, or element description files. To create a new text file (and open a new text window), select **File > New > Text**. LVS will create a new text window with the default name **LVS1**. When you save the text file, you can either use the default name or supply your own, but you must supply an appropriate filename extension, such as **.spc** or **.elm**. Subsequent new text windows will receive similar names by default—**LVS2**, **LVS3**, etc.

Using Go To

To speed searches in a text file, you can use **Edit > Go To**. Depending on the value you enter and your selection, LVS will move the cursor directly to the beginning of the specified line, column or bookmark.
Verification Window

LVS reports its progress and the results of a verification run in the verification window.

The verification results will appear in this window whenever you run a verification. You can also display the results of the last-performed verification by selecting View > Verification Window anytime after the initial verification run. If you select this command before running a verification, the window will be empty.

The verification window is divided into three sections, or panes. The uppermost section lists the input and output files used in the verification, and the current status of each file. A graphic indicator also displays the progress LVS has made in the current process.

The middle section of the verification window reports the result of a verification run, relevant notes, and any errors encountered during the verification. Below the text, a graphic indicator displays the progress LVS has made in processing the verification run as a whole.

The bottom section of the verification window contains an editable copy of the output file, which reports input files and options set in the setup window, plus verification progress statements written during the run. This section can be used to perform line edits on the output file using the following basic
text editing commands. Note that in the verification window these commands are only available via the keyboard.

**Ctrl+A**
- Select all.

**Ctrl+C**
- Copy to clipboard.

**Ctrl+F**
- Find.

**Ctrl+G**
- Go to line.

**Ctrl+S**
- Save to file.

**Ctrl+V**
- Paste from clipboard.

**Ctrl+X**
- Cut to clipboard.

**Ctrl+Home**
- Position cursor on first line.

**Ctrl+End**
- Position cursor on last line.

**Page Up**
- Scroll up through file.

**Page Down**
- Scroll down through file.

**Delete**
- Delete line.

In the verification window (“Verification Window” on page 990), the bottom section shows the derived from entries made in the setup window. This is used to initiate the verification run:

```plaintext
: lvs C:\Tanner\LEdit83\Samples\LVS\layout_resonator.spc
   C:\Tanner\LEdit83\Samples\LVS\schematic_resonator.spc
   -o C:\Tanner\LEdit83\Samples\LVS\resonator.out
   -l C:\Tanner\LEdit83\Samples\LVS\layout_resonator.lst
   -e C:\Tanner\LEdit83\Samples\LVS\resonator.elm
   -nrcl -y2 -fafpr
```

Beneath the , LVS displays the options chosen in the setup window:

**Engine configuration report:**
Consider Bulk nodes........................................ON
Consider Resistors as polarized elements........................OFF
Consider Capacitors as polarized elements........................OFF
Consider Inductors as polarized elements........................OFF
Merge series, parallel R and C, parallel MOSFETs.
  Delete shorted devices........................................ON
Merge series MOSFETs..............................................OFF
Find series MOSFETs that differ in order........................OFF
Fast Iteration..................................................OFF

Next, the verification window displays the progress statements written during the verification run:

```plaintext
Reading element definitions from file
   C:\Tanner\LEdit83\Samples\LVS\resonator.elm
Parsing file C:\Tanner\LEdit83\Samples\LVS\layout_resonator.spc...
Flattening netlist...
Parsing file C:\Tanner\LEdit83\Samples\LVS\schematic_resonator.spc...
```

For further information on LVS verification output, see the “LVS Output Tutorial” on page 1010.
You can rerun the verification whose results are reported in the verification window by clicking on the appropriate setup window and pressing F5.

**Verification Queue**

You can run several verifications consecutively by creating a verification queue. Once you have created such a queue, you can use it to run multiple verifications without additional user intervention.

To create a verification queue or add a setup to an existing queue, select **Verification > Add to Queue** or click the following toolbar button:

LVS will automatically add the active setup to the end of the existing queue.

To display the verification queue, select **Verification > Verification Queue (F8)** or click the **Open Queue Dialog ( )** button. LVS will display a dialog like this:

Each setup is listed in the **Setup Names** field. You can make changes to the existing queue via keyboard shortcuts or a toolbar within the dialog. The toolbar buttons, with their functions and corresponding keyboard shortcuts, are illustrated below:

Clicking **OK** saves any changes you have made to the verification queue and closes the dialog. LVS saves the most recent verification queue in the registry, allowing you to retrieve the current settings the next time LVS is launched.
Clicking **Run** initiates a verification for every setup listed in the verification queue. You can also initiate such a verification run with the verification queue closed by selecting **Verification > Start Verification Queue (F9)** or by clicking the **Run Queue** toolbar button:

When LVS has finished running the verification queue, it will display a dialog like the following:

![Queue Status Dialog](image)

**LVS Output File Format**

LVS writes the verification results to an output file with the extension `.lvs` (or `.out`). The LVS `.out` file is a text file that contains the used to initiate the verification run, verification options, progress statements written during the verification run, and the results of the verification itself.

In the following example, this is used to initiate the verification run:

```
: lvs layout_resonator.spc schematic_resonator.spc -o C:\test\resonator.lvs -l layout_resonator.lst -e \Petervpc\lvs v3.0 rel\Examples\resonator.elm -nrc1 -y2 -vfpar
```

Beneath the , LVS displays the options chosen in the setup window:

```
Engine configuration report:
Consider Bulk nodes...................................................ON
Consider Resistors as polarized elements..............................OFF
Consider Capacitors as polarized elements.............................OFF
Consider Inductors as polarized elements..............................OFF
Optimize shorted & parallel R, C, MOSFETs; series R and C...............OFF
Replace series MOSFETs................................................OFF
Fast Iteration................................................................OFF
```

Next, the verification window displays the progress statements written during the verification run:

```
Reading element definitions from file \Petervpc\lvs v3.0 rel\Examples\resonator.el
Parsing file layout_resonator.spc...
Not resolving subckt fspring
Not resolving subckt comb
Not resolving subckt fspring
Not resolving subckt comb
Not resolving subckt plate4
Parsing file schematic_resonator.spc...
```
Verification results can vary widely according to problems encountered during the verification run. For further information on LVS verification output, see the “LVS Output Tutorial” on page 1010.

**Node and Element List Format**

The node and element list .lst is an optionally generated output text file, that names matching and unresolved nodes and elements, by iteration, in the two compared netlists.

An *element* from one file cannot be equated with a *node* from the other file.

**Syntax**

For each iteration, nodes and elements are listed in the order:

- Resolved nodes
- Other (unresolved) nodes
- Resolved elements
- Other (unresolved) elements

Resolved nodes and elements are listed as follows:

```
member1a <=> member2a
member1b <=> member2b
... 
```

**member1** … The entry on the *left* in each line belongs to the **Layout netlist** specified in the **Run LVS** dialog.

**member2** … The entry on the *right* in each line belongs to the **Schematic netlist** specified in the **Run LVS** dialog.

Other (unresolved) nodes and elements are listed in single-column format.

**SPICE File Format**

By using the correct commands, an entire circuit and all contained devices can be described in SPICE format.

The maximum number of characters per line is 80. Statements extending over multiple lines must have the + continuation character in the first column of each line after the first.

Filenames are dependent on the operating system in use; the filename specified in an .include command must meet operating system requirements, including maximum length and special character requirements.

**Note:** File paths that contain spaces must be enclosed in single quotes.
Device Statements

Each device statement in a SPICE file begins with a key letter indicating the device type, followed by the unique name assigned to the specified element. The combination of key letter and name define a unique instance of a device. Device parameters are listed after the element name. They may include:

- Names of the nodes connected to device terminals.
- The name of a corresponding device model. Device models are declared using the .model command (see “.MODEL” on page 999).
- An electrical value such as capacitance, inductance, or resistance.
- An area scale factor, [AREA=]area or [A=]area in D, Q, J Z and M devices for compatibility with Calibre netlist extraction.
- Geometric parameters such as length and width.
- A multiplicity factor, [M=mult]. Multiplicity indicates the number of devices occurring in parallel; the default value of this parameter is 1.

The following table describes the correct syntax for device statements supported by LVS.

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>BJT (Q)</td>
<td>Qname collector base emitter [substrate] + model ([AREA=]area) [M=mult]</td>
</tr>
<tr>
<td>Capacitor (C)</td>
<td>Syntax 1</td>
</tr>
<tr>
<td></td>
<td>Syntax 2</td>
</tr>
</tbody>
</table>

The parameters collector, base, emitter, and substrate (optional) specify the nodes connected to each of these terminals.
The third syntax allows you to calculate capacitance using length and width parameters and an appropriate model statement:

```
Cname node1 node2 modelname L=length W=width
```

To use this syntax, you must specify a model that is declared in a model statement of the form:

```
.model modelname C [COX=val] [CAPSW=val][params...]
```

where `C` is the device type and `COX` and `CAPSW` are used to calculate capacitance from the length and width values. The calculated capacitance is equal to `COX*L*W+2*(L+W)*CAPSW`. At least one of `COX` or `CAPSW` must be present. Additional model parameters are optional.

**Diode (D)**

```
Dname node1 node2 model [[AREA=]area] + [M=mult]
```

The node names `node1` and `node2` represent the positive and negative terminals, respectively, of the diode.

**Inductor (L)**

```
Lname node1 node2 [model] [L=]l [M=mult]
```

The node names `node1` and `node2` represent the positive and negative terminals, respectively, of the inductor. The inductance is specified by the parameter `L=l`.

**JFET (J)**

```
Jname drain gate source [bulk] model + [[AREA=]area] [M=mult]
```

The parameters `drain`, `gate`, `source`, and `bulk` (optional) specify the nodes connected to each of these terminals.

**MESFET (Z)**

```
Zname drain gate source [bulk] model + [[AREA=]area] [M=mult]
```

The parameters `drain`, `gate`, `source`, and `bulk` (optional) specify the nodes connected to each of these terminals. (LVS also supports the `B` key letter in MESFET statements.)

**MOSFET (M)**

```
Mname drain gate source [bulk] model [L=length] + [[W=]width] [AD=ad] [PD=pd] [AS=as] [PS=ps] + [NRD=nrd] [NRS=nrs] [NRG=nrg] [NRB=nrb] + [M=mult]
```

The parameters `drain`, `gate`, `source`, and `bulk` (optional) specify the nodes connected to each of these terminals. In addition, MOSFETs use the following optional parameters:

- `length, width`—Channel length and width.
- `ad, pd`—Drain area and perimeter.
- `as, ps`—Source area and perimeter.
- `nrd, nrs, nrg, nrb`—Number of squares of diffusion for drain, source, gate, and bulk.

**Resistor (R)**

There are two syntax options for a resistor; the choice depends on whether a sheet resistance value is required to calculate the resistance. In both cases, `node1` and `node2` are the nodes spanned by the resistor.
Chapter 28: Introduction to LVS User Interface

Syntax 1

The first syntax requires that you specify the resistance, \( r \).

\[
\text{R name node1 node2 [model] [R=r [M=mult]]}
\]

The use of a resistor model is optional. However, if you do include a model name, it must be declared in a model statement of the form:

\[
.model \text{ modelname [R=[RSH=val]]}
\]

where the optional parameters \( R \) and \( RSH \) specify the model type and sheet resistance, respectively.

Syntax 2

You can optionally specify resistor length and width instead of the resistance (\( r \)).

\[
\text{R name node1 node2 model L=length W=width [M=mult]}
\]

To use length and width parameters, you must specify a resistor model that corresponds to a model statement of the form:

\[
.model \text{ modelname R RSH=val}
\]

In this syntax, the device type (\( R \)) and sheet resistance (\( RSH \)) are required parameters.

Transmission Line (T)

\[
\text{T name n1a n1b n2a n2b [Z0=z] [TD=d] [f=F] [nl=N]}
\]

Terminals \( n1a (+) \) and \( n1b (-) \) are at one end of the transmission line, and \( n2a (+) \) and \( n2b (-) \) are at the opposite end. Additional transmission line parameters are:

- \( z \)—Impedance (ohms).
- \( d \)—Transmission delay (seconds). The delay may instead be specified indirectly from \( f \) and \( n \).
- \( F \)—Line frequency (Hertz).
- \( N \)—Normalized number of wavelengths. The transmission delay \( (d) \) is equal to \( n/f \).

Note:

Only the SPICE devices and parameters relevant to LVS are listed here. A complete list of available devices can be found in the T-Spice or H-SPICE user guides.

Subcircuit Instances

Subcircuit instance statements have the following syntax:

T-Spice / H-SPICE:

\[
X \text{name node1 [node2 ...] cname [par1=p1 par2=p2 ...]}
\]

P-SPICE:

\[
X \text{name node1 [node2 ...] cname \text{PARAMS:}[par1=p1 par2=p2 ...]}
\]

For Extract, the syntax varies as follows:
Xiname node1 [node2 ...] cname [AREA=layerArea/areaVal]
[AREA_pinName=pin1Area/areaVal]
[AREA_pinName=pin2Area/areaVal] ...

iname A unique instance name.
node1 node2 Node names. There must be as many node names listed as there are in the subcircuit definition.
par1 par2 The list of available parameters is determined by the subcircuit definition. Parameter assignments are optional on a subcircuit instance statement; parameters may be listed in any order. If a parameter’s value is not specified in the instance statement, its value is taken from the default assigned in the subcircuit definition statement. The multiplicity parameter (M=m) is implicitly defined and can also be listed.
cname The subcircuit name (from the definition statement).

For example (in T-Spice or H-SPICE):

X123 N125 N253 N74 myCircuit AREA=100 Q=42 E=17

or (in P-SPICE):

X123 N125 N253 N74 myCircuit PARAMS: AREA=100 Q=42 E=17

instantiates a previously defined subcircuit called myCircuit. Its unique name, to distinguish it from other instances of the same subcircuit, is X123. It has three pins, connected to nodes N125, N253 and N74. It also has three parameters: AREA, Q, and E. The definition for this subcircuit is given as an example below.

For Extract

X123 N125 N253 N74 myCircuit AREA=100 AREA_Pin1=15

defines an instance X123 of a subcircuit called myCircuit. It has three pins, connected to nodes N125, N253, and N74.

Subcircuit Definitions

Subcircuit definition statements have the following syntax:

T-Spice / H-SPICE:

.SUBCKT name pin1 [pin2 ...] [par1=val1 par2=val2 ...]
<subcircuit definition>
.ENTS [name]

P-SPICE:

.SUBCKT name pin1 [pin2 ...] PARAMS:[par1=val1 par2=val2 ...]
<subcircuit definition>
.ENDS [name]

**name**
The name, or type, of the circuit.

**pin1 pin2**
The pins (inputs and outputs) to the circuit.

**par1 par2**
An optional list defining the parameters whose values must be known when the subcircuit is instantiated. A value given for a parameter here is its default, to be assumed if the parameter is not assigned a value on the subcircuit instance statement. The multiplicity (M=m) parameter may not be included here.

The last line of the subcircuit definition can optionally contain the same subcircuit name used in the first line of the definition (for example, `.ends MYCIRCUIT`).

In between the first (.subckt) and last (.ends) lines are any number of other SPICE commands and statements (except subcircuit instance and model commands).

If the subcircuit is empty, it must be defined as an element in the special element file to be used with LVS.

Duplicate subcircuits are ignored, with a warning.

**SPICE Statements**

**.INCLUDE**
A SPICE file can include the contents of other SPICE files with the `.include` command.

The `.include` command has the following syntax:

```
.include 'filename'
```

The `filename` can be the name of any other SPICE file, and can include drive and path information, if needed. The filename must be contained within single quotes. Care should be taken to ensure that inclusion commands do not involve logical loops (for example, `fileA` including `fileB`, which itself includes `fileA`).

**.MODEL**
The `.model` command defines a model to be used in device statements. It can appear anywhere in the SPICE file, even after the specified model is mentioned in an element statement.

The `.model` command has the following syntax:

```
.model name type [par1=p1 par2=p2 ...]
```

**name**
The name of the model.

**type**
One of the following: C (capacitor); R (resistor); L (inductor); D (diode); NPN or PNP (BJT); NJF or PJF (JFET); NMOS or PMOS (MOSFET); NMF or PMF (MESFET).

In LVS, this parameter is ignored and can be left off.
The parameters for the model are listed after the model type, and are specific to the model type. The set of parameters determines the SPICE behavior of the model. However, for extraction and netlist comparison, the parameters are not used and can be left off.

For example:

```
.model mydevice nmos
```

specifies an NMOS MOSFET device named `mydevice`. The model name can be used in device statements such as the following:

```
m123 42 51 7 mydevice l=2 w=28
```

This defines an NMOS transistor with the unique name of `M123`. Its drain is connected to node `42`, its gate to node `51`, and its source to node `7`. It has a length of `2` and a width of `28`.

**.MALIAS**

The `.malias` command assigns an alias to a model name which was defined in a `.model` command so that device references to model names may use either the original model name or the alias name.

Alias name assignments may also refer to binned model base names, with declarations for `modelname.1`, `modelname.2`, etc., combined into a single `modelname` and alias reference. For example, given binned models `nch.1`, `nch.2`, `nch.3`, etc. you may declare an alias name “`nmosmodel`” for the model name “`nch`.”

The `.malias` command has the following syntax:

```
.malias ModelName=AliasName1 [ AliasName2 [ AliasName3 [ ... ] ] ]
```

For example:

```
.model nch.1 nmos level=49 ...
.model nch.2 nmos level=49 ...
.malias nch=nmosmodel
```

or:

```
.model dio d ...
.malias dio=zenerdiode
```

**Auto-Declaration of Models in LVS**

When a device statement includes a model name, LVS looks for a corresponding `.model` statement in the netlist. If the `.model` statement is missing, LVS can automatically add a simple model declaration without any parameters (e.g., `.model mydevice`). LVS will auto-declare models for diode (D), JFET (J), GaAsFET (B), BJT (Q), MOSFET (M), and MESFET (Z) device statements.

This is necessary to resolve a fundamental ambiguity in SPICE. BJT's and other devices can have three terminals or four; they also need a model. In the case of a three terminal device, LVS can't tell if the fourth pin is a fourth terminal or a model if it has not yet encountered either that terminal nor that model before. When it has encountered a previous definition of that net on another device, LVS can resolve this ambiguity. Otherwise it will auto-declare the model.
Auto-declared model names appear as warnings in the LVS output. For example, auto-declaration of the MOSFET model M1 would result in the following warning:

```
Warning: test1.sp(4): Implicit .model definition M1
```

When possible, it is best to export model declarations of mosfet and bipolars, etc. from your netlist, as they are necessary for L-Edit, yet do not cause any problem for LVS.

**Note:**

LVS does *not* auto-declare models for resistors (R), capacitors (C), or inductors (I). In these device statements, using a model name that is not declared with `.model` will result in an error.

### .GLOBAL

The `.global` command declares certain nodes as global throughout the SPICE file. It is used in SPICE files containing subcircuits in order to make certain signals available to the subcircuit without explicitly having to declare them in the subcircuit definition. Typical global nodes might include clocks, power, the data bus, etc.

The `.global` command has the following syntax:

```
.global node1 node2 ...
```

For example:

```
.global clock data1 data2 data3 data4
```

Without `.global`, the `clock` node would be considered local to the subcircuit in which it is defined and distinct from any other node called `clock` outside of the subcircuit. With `.global` however, every node called `clock` inside or outside of a subcircuit is equivalent.

If the `-pspice` option is specified on the , LVS recognizes PSPICE global nodes.

### .OPTION

The `.option` command has the following syntax:

```
.option [scale=s][parhier=local|global|tspice]
```

LVS scales all geometric parameters stated in device statements by the given value `s`. For example, a scale value of 2 doubles the length and width parameters of all MOSFETs, and squares the area parameter on all diodes, BJTs, etc.

The parhier option determines the precedence with which parameter values are applied. If `parhier=global`, a higher level parameter definition overrides a parameter defined in lower level of the design hierarchy. If `parhier=local`, the innermost scope takes precedence. Setting `parhier=tspice` is similar to `parhier=local`, except that in a subcircuit, the parameter value defined on the `.subckt` line has higher priority than a `.param` within the subckt definition. The default is `parhier=global`.

For example:

```
.option scale=2
.option scale=100m
```
.PARAM

The .param command defines symbolic parameter values so that they can be used anywhere that a parameter value is called for.

The .param command has the following syntax:

```
.param symbol1=n1 symbol2=n2 ...
```

For example:

```
.param cap100=100pf tranlen=2u tranwid=28u
```

specifies parameter values that could then be used in statements such as:

```
c123 12 56 c=cap100
m43 24 54 300 nmos l=tranlen w=tranwid
.subckt mycircuit in out reset c=cap100
```

Parameters can also be defined in terms of arithmetic expressions.

For example:

```
.param resistance=10
R1 1 2 R='resistance*2'
```

would create a 20 ohm resistor.

Subcircuits can have parameters when defined, for example

```
.subckt res a b resistance=20
```

Subcircuit calls can also have parameters, for example

```
x1 1 2 res resistance=30
```

The order of precedence of parameter evaluation depends on the .option parhier. In the default case (.option parhier=global), the outermost definition takes precedence.

Consider the following example:

```
.param resistance=A
.subckt res a b resistance=G
.param resistance=F
R1 a b R='resistance'
.ends
.subckt r a b resistance=D
.param resistance=C
x1 a b res resistance=E
.subckt
X1 1 2 r resistance=B
```

The order of precedence, from high to low, is (A, B, C, D, E, F, G).

If .option parhier=local, the precedence becomes (E, F, G, B, C, D, A).
if `.option parhier=tspice`, the order is (E, F, G, B, C, D, A)

In all cases, parameters defined on the subcircuit call have higher priority than those defined inside the subckt using `.param`, which in turn have higher priority than those defined on the `.subckt` definition line.

.END

A SPICE file is terminated with an `.end` command on the last line of the file. Anything following this command is ignored.

The `.end` command has the following syntax:

```
.end
```

**Parameters**

Parameter values can take many forms. Some examples are:

```
area=10     l=.001     r=3.4e-3     c=cap100
```

In the last example, `cap100` must have previously been defined by a `.param` command. None of the examples specify units, so default units are assumed (ohms for resistance, farads for capacitance, meters for length, square meters for area, and so on).

Numbers can be followed by metric abbreviations indicating order of magnitude. The base units (s, v, a, f, h) are implicit from the context; any characters following the metric abbreviation are ignored. For example, the following expressions can all specify a capacitance of 10.2 picofarads:

```
C=10.2pF
C=10.2P
C=10.2pxyz
```

Acceptable metric prefix abbreviations are shown in the following table.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Prefix</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>t or T</td>
<td>tera-</td>
<td>$10^{12}$</td>
</tr>
<tr>
<td>g or G</td>
<td>giga-</td>
<td>$10^{9}$</td>
</tr>
<tr>
<td>meg or MEG</td>
<td>mega-</td>
<td>$10^{6}$</td>
</tr>
<tr>
<td>k or K</td>
<td>kilo-</td>
<td>$10^{3}$</td>
</tr>
<tr>
<td>m or M</td>
<td>milli-</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>u or U</td>
<td>micro-</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>n or N</td>
<td>nano-</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>p or P</td>
<td>pico-</td>
<td>$10^{-12}$</td>
</tr>
<tr>
<td>f or F</td>
<td>femto-</td>
<td>$10^{-15}$</td>
</tr>
</tbody>
</table>

A commonly used abbreviation is the unit mil (or MIL), representing $10^{-3}$ inch.

Parameters listed more than once on the same statement assume the first assigned value. For example:
defines a 10 picofarad capacitor, not a 20 microfarad one.

Comments

Comment lines are designated by an asterisk (*) in the first column. In-line comments are designated by a dollar sign ($) for T-Spice and H-Spice format netlists, or by a semi-colon (;) for P-Spice format netlists. All text following the inline comment character up to the end of the line on which it is found is ignored.

The first line of any SPICE file is always considered a comment, even without comment delimiters.

CDL Files

Circuit Description Language (CDL) files are regular SPICE files, with a few minor syntax changes:

- Subcircuit calls contain a “/” before the name of the subcircuit being instanced.
- Resistors and capacitors contain optional models and physical sizes. For example: “R11 net1 net2 1k $[B] $W=6u” or “C11 net1 net2 5p $[F]”.
- A three terminal (bulk) resistor device is available. LVS maps these devices into subcircuits. For example, the resistor “R11 net1 net2 1k $SUB=VCC $[B] $W=6u” becomes equivalent to the subcircuit call: “XR11 net1 net2 VCC B R=1k W=6u”

Restrictions and Extensions

There are many varieties of SPICE, but only those conforming to the syntax described above are supported for the purposes of netlist comparison and layout extraction.

Unsupported parameters, if present, are ignored. Default values for parameters are not supported, except for the multiplicity parameter and subcircuit parameters. If a parameter is not specified but is needed for a computation, an error will occur and a warning message will be displayed.

Subcircuit definitions may not have model statements or other subcircuit definitions within them.

The .bulk command is not supported.

Only certain model types are supported (see "MODEL" (page 999)); others are ignored. Parameters for model statements are not utilized. Any parameters listed in a model statement will be ignored.

For netlist comparison, the SPICE format supported by LVS contains an additional feature to enable comparison of non-standard elements. This is accomplished by defining the additional elements in an element description file, and then using the devices in the same manner as you would use a subcircuit element.

Using LVS in Batch Mode

Using LVS in batch mode provides another way to run multiple verifications consecutively. To run LVS in batch mode, you must first create a batch file. A batch file is a text file containing one or more command-line invocations of LVS and appropriate setup information—input files and verification options—for each verification run.
Creating a Batch File

To create a batch file, select File > Export Batch File (Ctrl+E) with a setup window active. LVS will display the Export Batch File as dialog. Navigate to the correct directory if necessary and enter the name of the batch file in the File name field, then click Save. LVS will create a batch file with a single invocation of LVS and command-line options derived from the parameters set in the active setup window. A simple batch file will contain text like the following:

```
start /w lvs "C:\LVS\Examples\Ex2_1.spc" "C:\LVS\Examples\Ex2_2.spc" -o "C:\LVS\Examples\Ex2.out" -nrcl -c1245 -dv5.000 -dg0.010 -vfa -q
```

For further information on batch-file syntax, see “Batch-File Syntax” on page 1030. For further information on command-line options, see “Options” on page 1031.

To add verifications to an existing batch file, select it in the Export Batch File as dialog and click Save. LVS will prompt you with the following dialog:

To add to the current batch file, click Append To. LVS will append an appropriate to the existing batch file. If you click Overwrite, LVS will overwrite the existing batch file with a new using setup information derived from the current setup window. You can further develop a batch file by editing it in the LVS text window. For example, you can change the names and paths of the SPICE files used in the verification, or set different command-line options.

To run an LVS batch file, double-click it in Windows Explorer or invoke it from a DOS command prompt. Note that LVS does not support launching of multiple instances—to run a batch file, you must first exit LVS for Windows. For further information, see “Running a Batch File” on page 1030.
Tutorial

This brief tutorial will teach you how to perform the following basic tasks in LVS:

- Set up and run a verification
- Create a verification queue
- Create and run a verification batch file

For information on interpreting LVS output and resolving verification problems, see the “LVS Output Tutorial” on page 1010.

Creating a Verification Setup

In the following steps, you will learn to create and run an LVS verification database (.vdb) file. The file you will create is also provided as \C:\{install_dir}\Samples\LVS\ex1.vdb.

- Start LVS, if it is not already running.
- Select File > New or click the New button in the toolbar.
- In the New dialog, select LVS Setup as the file type, and click OK.
- LVS will display a setup window. In the File tab, type the names of the two netlists you wish to compare. You can also use the Browse buttons to navigate to these files. For this tutorial, use the files ex1_1.spc as the layout netlist and ex1_2.spc as the schematic netlist. Both of these files are located in \C:\install_dir\Samples\LVS.
- Check the Output file option and enter the name of the output file in the corresponding field. For this tutorial, use \C:\install_dir\Samples\LVS\ex1.out.
- Check the option Overwrite existing output files so that LVS will overwrite this output file on subsequent verification runs without prompting you.
- On the Options tab, check the following options:
  - Consider MOSFET bulk nodes (substrate) during iteration matching
  - Consider resistors as polarized elements
  - Consider capacitors as polarized elements
  - Consider inductors as polarized elements

- On the Device Parameters tab, check Resistance, capacitance, and inductance values. Enter a Maximum element-value tolerance of 5.000.
- On the Performance tab, select Normal iteration: consider fanout and element type.
- On the Verbosity Level tab, check the following options for both Output file and Screen:
  - Show fragmented classes
  - Show automorph classes
  - Show series MOSFETs that differ in order
Save your setup as **ex1.vdb**, or another name of your choice.

Select **Verification > Run** (F5) or click the **Run Verification** ( ) button on the toolbar.

This verification run will produce an element/node count mismatch. LVS will prompt you for permission to continue iteration. Click **Yes**.

LVS will complete the verification run and display the following results in the verification window:

0 perfectly matched element class(es)
1 fragmented element class(es)
3 perfectly matched node class(es)
12 fragmented node class(es)

Circuits are not equal.

This verification run results in one fragmented element class and 12 fragmented node classes. It is not necessary to resolve these fragmented classes now, but the correct procedure is described in “Resolving a Fragmented Class” on page 1016.

Now create another verification setup using the following input files and parameters. Save your setup file as **ex2.vdb** (or use the setup file supplied with LVS).

- On the **File** tab, select `C:\install_dir\Samples\LVS\ex1_2.spc` as the layout netlist file and `C:\install_dir\Samples\LVS\ex2_1.spc` as the schematic netlist file. Check the **Output file** option and name the output file `C:\install_dir\Samples\LVS\ex2.out`.
- On the **Options** tab, check the following options:
  - Consider MOSFET bulk nodes (substrate) during iteration matching
  - Consider resistors as polarized elements
  - Consider capacitors as polarized elements
  - Consider inductors as polarized elements
- On the **Device Parameters** tab, check the following options:
  - Resistance, capacitance, and inductance values. Then enter a Maximum element-value tolerance of 5.000.
  - Areas (in B, D, J and Q Elements). Then enter a Maximum geometric-value tolerance of 5.000.
- On the **Performance** tab, select Normal iteration: consider fanout and element type.
- On the **Verbosity Level** tab, check the following options for both **Output file** and **Screen**:
  - Show fragmented classes
  - Show automorph classes
  - Show series MOSFETs that differ in order

Select **Verification > Run** (F5) or click the **Run Verification** ( ) button on the toolbar. LVS will complete the verification run and display the following results in the verification window:

32 perfectly matched element class(es)
18 perfectly matched node class(es)
4 fragmented node class(es)
Circuits are not equal.

When you are finished viewing the results, close the verification window.

Creating a Verification Queue

In the following steps, you will learn how to create and run a verification queue.

Select `ex2.vdb` as the active setup window.

Select `Verification > Add to Queue (F7)` or click the `Add to queue` button ( ) on the toolbar.

Select `Verification > Verification Queue (F8)` or click the `Open Queue Dialog` button ( ) on the toolbar. LVS will display the `Verification Queue` dialog with the file `ex2.vdb` listed in the queue.

Within the verification queue dialog, click the `Add setup to queue` button ( ) or type the shortcut `Ctrl+O`. LVS will open a standard Windows file browser.

Select the file `C:\install_dir\Samples\LVS\ex1.vdb` and click `Open`. LVS will add this setup file to the verification queue.

Repeat the last two steps and add file `ex3_1.vdb` to the queue.

If desired, rearrange the setups in the queue, using the arrow keys or appropriate buttons ( and ) in the `Verification Queue` dialog.

To run the verification, click `Run` in the `Verification Queue` dialog, or click `OK` to close the dialog, then select `Verification > Start Verification Queue (F9)`. You can also click the `Run Queue` ( ) button in the application toolbar.

LVS will run the verifications in the order listed and report the results in the `Queue Status` dialog. You should see the following results (in the order you specified):

```
C:\Tanner\LEdit84\Samples\LVS\ex2.vdbCircuits are not equal!
C:\Tanner\LEdit84\Samples\LVS\ex1.vdbCircuits are not equal!
C:\Tanner\LEdit84\Samples\LVS\ex3_1.vdbCircuits are equal!
```

Close the `Verification Queue` dialog, if necessary.

In the next section, you will learn how to create and run a verification batch file.

Running LVS in Batch Mode

Select `ex1.vdb` as the active setup window.

Select `File > Export Batch File (Ctrl+E)`.

LVS will display the `Export Batch File as` dialog. Select `ex1.bat` or type it in the `File name` field. Click `Save`.

Select `ex2.vdb` as the active setup window.

Select `File > Export Batch File (Ctrl+E)`. In the `Export Batch File as` dialog, type `ex1.bat` in the `File name` field and click `OK`.
LVS will prompt you with the dialog **Batch file already exists.** Click the **Append To** button.

Select **ex3_1.vdb** as the active setup window and repeat the last two steps to add this setup to the batch file.

Exit LVS and open a DOS window. Navigate to the LVS installation directory, then type **Samples\LVS\ex1.bat.** For default installations, the path should read:

```
C:\Tanner\LEdit84> Samples\LVS\ex1.bat
```

LVS will run the verifications listed in the batch file and write the results to the specified output file(s). You can view the results using the LVS text window or any other text editor.
29 LVS Output Tutorial

Introduction

LVS generates certain output upon execution, depending on which options are checked on the “Setup—Output” (page 978) tab in the setup window. In the following examples, the .vdb files are opened in LVS and run with a single invocation. If run in batch mode, user interactions, defaults, and generated output may differ.

All files used in this chapter are available in the \install_dir\Samples\LVS\ directory.

Parsing Information

LVS first reports parsing information about both of the netlists.

Following is an example of the output ex1.out, generated by LVS for ex1.vdb. The options for Show Fragmented Classes, Show Automorph Classes, and Show Detailed Processing Information are checked in both Output file and Screen columns, so all information written to the output file is also written to the verification window.

Engine configuration report:
Consider Bulk nodes.................................ON
Consider Resistors as polarized elements............ON
Consider Capacitors as polarized elements...........ON
Consider Inductors as polarized elements............ON
Optimize shorted & parallel R, C, MOSFETs;
series R and C........................................OFF
Replace series MOSFETs..............................OFF
Fast Iteration.....................................OFF
Parameter comparison threshold for CAPACITORS.......5%
Parameter comparison threshold for INDUCTORS........5%
Parameter comparison threshold for RESISTORS........5%

Parsing file ex1_1.spc...
Including file MODELS.SPC
Flattening network...
Parsing file ex1_2.spc...
Including file MODELS.SPC
Flattening network...

<table>
<thead>
<tr>
<th>Device</th>
<th>ex1_1.spc</th>
<th>ex1_2.spc</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>M_NSS</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>M_PSS</td>
<td>26</td>
<td>24</td>
<td>MISMATCH (2)</td>
</tr>
<tr>
<td>Total element</td>
<td>50</td>
<td>48</td>
<td>MISMATCH (2)</td>
</tr>
<tr>
<td>Total nodes</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>
LVS begins the parsing information by stating which file was read and listing any files included in this file. After the network is flattened, the program reports the kinds of elements found and the number of nodes found. The program concatenates the device type with the model name to make up the element type, as in M_NSS. This helps distinguish between elements of the same device type that use different models.

If the program detects a numerical difference between the number of different elements and the number of nodes between the two files, it poses a question identifying the problem and requiring instructions to continue or exit. If the **Continue processing when mismatch in node or element count found** option on the “Setup—Performance” (page 988) tab in the setup window is checked, or if LVS is invoked from batch mode, the program does not stop at this point and continues on to the iteration.

**Parameter Matching Example**

Following is an example of the output **ex2.out**, generated by LVS for **ex2.vdb**. In this example, LVS is instructed to use parameter matching (**Resistance, Capacitance and Inductance values** for R, C, and L Elements and **Lengths and Widths** for MOSFET Elements on the “Setup—Device Parameters” (page 980) tab), in addition to the topological characteristics. (See “Parameter Matching” on page 1026 for more information on this method.)

Iterating...
5% done
10% done
...
25% done

Warning: Parametric mismatch between elements
ex1_2.spc: M346:L=2 W=5   (Not all decimals shown)
ex2_1.spc: M6(XSUBA/X34):L=2 W=6   (Not all decimals shown)

Warning: Parametric mismatch between elements
ex1_2.spc: M6:L=2 W=5   (Not all decimals shown)
ex2_1.spc: M6(XSUBA/X33):L=2 W=6   (Not all decimals shown)

50% done
...

In this example, during the iteration matching process LVS encountered elements that are topologically matched but are parametrically different. LVS reports this as a warning, but it does not affect the iteration process.

**Automorph Class Example**

When LVS finishes iteration, it reports any automorphism or fragmentation that occurs. Automorph classes are listed first.

Following is an example of the output **ex3.out**, generated by LVS for **ex3_1.vdb**.

**************************************** REPORTING AUTOMORPHISM ***************

Report of elements:
****************************************
Automorph class of elements
Ex3_1.spc M1(XSUBA/X34)_2fanout: BULK = 41D/S ( 3, 7) G = 7
Ex3_1.spc M1(XSUBA/X34)_1fanout: BULK = 41D/S ( 3, 7) G = 7
Ex3_2.spc M350 fanout: BULK = 41 D/S ( 3, 7) G = 7
Ex3_2.spc M341 fanout: BULK = 41 D/S ( 3, 7) G = 7
---------------------------------------------------
Automorph class of elements
...
...
---------------------------------------------------
Automorph class of elements
Ex3_1.spc: 4 element(s)
Ex3_2.spc: 4 element(s)
Ex3_1.spc C1(XSUBA/X34)_2 fanout: POS/NEG ( 35, 41)
Ex3_1.spc C1(XSUBA/X34)_1 fanout: POS/NEG ( 35, 41)
Ex3_1.spc C1(XSUBA/X33)_2 fanout: POS/NEG ( 35, 41)
Ex3_1.spc C1(XSUBA/X33)_1 fanout: POS/NEG ( 35, 41)
Ex3_2.spc C1_4 fanout: POS/NEG ( 35, 41)
Ex3_2.spc C1_3 fanout: POS/NEG ( 35, 41)
Ex3_2.spc C1_2 fanout: POS/NEG ( 35, 41)
Ex3_2.spc C1_1 fanout: POS/NEG ( 35, 41)
---------------------------------------------------
************************ ITERATION SUMMARY *********************
32 perfectly matched element class(es)
7 automorph element class(es)
20 perfectly matched node class(es)

Each element is listed on a separate line. For each element, the netlist file of origin is listed first (Ex3_1.spc) followed by the unique name of the element (M1(XSUBA/X34)_2). Hierarchical information is provided within the parentheses as well as with the appended number at the end. Whenever LVS encounters a line such as M1 N39 N23 N25 VDD PSS L=2 W=5 M=2 (line 14 of file Ex3_1.spc), the multiplier M=2 causes LVS to create several identical copies of an element. In this case, two copies are made. To distinguish between them, LVS appends a number at the end of each constructed element name.

The fanout tells what the “electrical” fanout of each terminal is. This describes the number of elements each terminal of the element is connected to, not counting the one obvious connection to itself. For example, element M1(XSUBA/X34)_2 has 41 elements connected to its bulk terminal, three elements to either its drain or source terminal, seven elements to the other of drain and source, and seven elements to its gate terminal. (LVS cannot distinguish between source and drain for FET transistors since there is no topological distinction between them.) All elements within a single automorph class have exactly the same fanout.

Since the Detailed Trial Matching to resolve Automorph Classes option was not checked in this example (see “Detailed Trial Matching” on page 1025), an alert box appears asking if LVS should use detailed trial matching to try and match the elements and nodes of the automorph classes. For the file ex3_1.vdb, detailed trial matching results in the conclusion that the circuits are equal. The output from detailed trial matching for this example is discussed on page 2-1013.

During detailed trial matching, LVS attempts to pair up the members of automorph classes. In most cases, this resolves the automorphism and the circuits are found equal. Occasionally the trial matching results in fragmentation; however, in the case of attempting to resolve an automorph class, it does not mean the two circuits are different. It only means that the particular trial matches did not resolve the particular automorph class.
Resolving Fragmentation of an Automorph Class

When detailed trial matching causes fragmentation of an automorph class, there are usually two reasons for this result:

- The trial-matching algorithm is not able to solve the problem. In this case you should provide a prematch file for sets of nodes and elements, giving LVS a “head start” on matching elements. (See “Preiteration Matching” on page 1025 for more information on prematch files.)
- There is, in fact, a difference between the two files being compared.

In either case, the way to move towards a solution at this point is to use a prematch file. Rerun LVS with the **Show Detailed Processing Information** option on the **“Setup—Output”** tab in the setup window checked. This option provides details about the various nodes and elements the detailed trial matching procedure matches. In most cases the last matched pair of elements or nodes causes the fragmentation. You will need to look back in your output and find a different match in the automorph class from which this last pair comes.

For example, if the last matched pair before fragmentation was:

Matched Elements C1(XSUBA/X34)_1 and C1_1

it means that LVS attempted to match element **C1(XSUBA/X34)_1** with element **C1_1**, and the match failed. Therefore, you should attempt to match a different element with **C1(XSUBA/X34)_1**, and a possible line in your prematch file would be:

**C1(XSUBA/X34)_1 C1_4**

The order in which elements and nodes are specified in a prematch file has to reflect the order in which the respective netlist files are specified on the.

With this line in the prematch file, rerun LVS again. Include the prematch file on the **“Setup—Input”** tab in the setup window.

If the problem of fragmentation recurs, then repeat the process above and add more lines of information to the prematch file.

If the two files being compared are equal, a few lines in the prematch file are all that is needed. If there is a discrepancy between the two input files, you will need to continue adding information to the prematch file until you are able to determine what the discrepancy is, using the methods of reasoning outlined above.

For the file **ex3_1.vdb**, detailed trial matching results in the following output:

....

*************** POST ITERATION MATCHING ***************
Doing detailed trial matching... Step 1 (Match by parameters)

*************** ITERATING ***************
Doing detailed trial matching... Step 2 (Random matches)
Matched Elements C1(XSUBA/X34)_2 and C1_4
Matched Elements C1(XSUBA/X33)_1 and C1_1
Matched Elements C1(XSUBA/X34)_1 and C1_3
90% done
Matched Elements M10(XSUBA/X34)_1 and M3410
Matched Elements M10(XSUBA/X33)_1 and M10
Matched Elements M19(XSUBA/X33)_1 and M19
95% done
Matched Elements M1 (XSUBA/X33)_2 and M3
Matched Elements M19(XSUBA/X34)_1 and M3419
Matched Elements M1(XSUBA/X34)_2 and M350
100% done

************************** FINAL RESULT **************************

Circuits are equal.

Besides providing the information that the circuits are equal, the output also gives information about what elements were matched during the trial matching process. This information can be turned into a prematch file to speed up the iteration process next time LVS is run. The above output is listed in the prematch file as:

M1(XSUBA/X34)_2   M350
M19(XSUBA/X34)_1 M3419
M1(XSUBA/X33)_2   M3
M19(XSUBA/X33)_1 M19
M10(XSUBA/X33)_1 M10
M10(XSUBA/X34)_1 M3410
C1(XSUBA/X33)_1   C1_1
C1(XSUBA/X34)_1   C1_3
C1(XSUBA/X34)_2   C1_4

The name of the prematch file is **ex3_1.pre**, and it is entered in the **Prematch File** field on the “Setup—Input” (page 977) tab in the setup window in **ex3_2.vdb**. Following is the output generated by LVS for **ex3_2.vdb**.

....
....
Processing file C:\Tanner\LEdit83\Samples\LVS\Ex3_1.pre of matched elements/nodes
Matched Elements M1 (XSUBA/X34)_2 and M350
Matched Elements M19(XSUBA/X34)_1 and M3419
Matched Elements M1 (XSUBA/X33)_2 and M3
Matched Elements M19(XSUBA/X33)_1 and M19
Matched Elements M10(XSUBA/X33)_1 and M10
Matched Elements M10(XSUBA/X34)_1 and M3410
Matched Elements C1(XSUBA/X33)_1 and C1_1
Matched Elements C1(XSUBA/X34)_1 and C1_3
Matched Elements C1(XSUBA/X34)_2 and C1_4

************************** ITERATING **************************

Iterating...
....
....

************** REPORTING AUTOMORPHISM **************

Report of elements:

***************

Automorph class of elements
Ex3_1.spc C1(XSUBA/X33)_2 fanout: POS/NEG ( 35,  41)
Ex3_2.spc C1_2 fanout: POS/NEG ( 35,  41)

**************** ITERATION SUMMARY ******************

47 perfectly matched element class(es)
1 automorphed element class(es)
20 perfectly matched node class(es)

*************** POST ITERATION MATCHING ***********************
Doing detailed trial matching... Step 1 (Match by parameters)

*************** ITERATING ******************************
Doing detailed trial matching... Step 2 (Random matches)
Matched Elements C1(XSUBA/X33)_2 and C1_2

*************** FINAL RESULT ******************************
Circuits are equal.

LVS matched nine out of ten of the original automorph elements before the iteration process began. In the file **ex3_3.vdb**, the last element is added to the prematch file **ex3_2.pre**. When LVS is run, all automorph classes are matched.

In this example only automorph classes of elements have been shown. Automorph classes of nodes are addressed in a similar fashion.

## Fragmented Class Example

Fragmentation can occur either during the regular iterative process or during the detailed trial matching procedure in the case of previous automorphism. Following is an example of the output (**ex4.out**) generated by LVS for **ex4.vdb**, including fragmented classes.

```plaintext
...  ...
...
45% done
50% done
55% done

*************** REPORTING FRAGMENTATION ***********************

Report of elements:
Fragmented class of elements
Ex1_1.spc: 6 element(s)
Ex1_2.spc: 4 element(s)
...
...

Report of nodes:
Report of nodes:
Fragmented class of nodes
Ex1_2.spc16  connected to1 M_NSS_D/S2 M_NSS_G
4 M_PSS_D/S4 M_PSS_G
--------------------------------------------------------------------------------

Fragmented class of nodes
Ex1_2.spc3  connected to3 M_NSS_D/S1 M_NSS_G
3 M_PSS_D/S1 M_PSS_G
Ex1_2.spc15  connected to3 M_NSS_D/S1 M_NSS_G
3 M_PSS_D/S1 M_PSS_G
--------------------------------------------------------------------------------
```
Fragmented class of nodes
Ex1_2.spc7 connected to M_NSS_G M_PSS_G

Fragmented class of nodes
Ex1_2.spc5 connected to M_NSS_D/S2 M_NSS_G
2 M_PSS_D/S4 M_PSS_G

Fragmented class of nodes
Ex1_2.spc4 connected to C_POS/NEG M_PSS_BULK
14 M_PSS_D/S

Fragmented class of nodes
Ex1_1.spc5 connected to M_NSS_D/S2 M_NSS_G
5 M_PSS_D/S4 M_PSS_G

Fragmented class of nodes
Ex1_1.spcN(XSUBA/X33/) connected to M_NSS_D/S
1 M_NSS_G
4 M_PSS_D/S1 M_PSS_G

Fragmented class of nodes
Ex1_1.spcVDD connected to C_POS/NEG26 M_PSS_BULK
14 M_PSS_D/S

0 perfectly matched element class(es)
1 fragmented element class(es)
4 perfectly matched node class(es)
10 fragmented node class(es)

Circuits are not equal.
**************

Fragmented element classes are listed first. In this example there are no fragmented element classes. Node classes are listed thereafter. The first class shows node 16 from file ex1_2.spc. The node is connected to one drain/source terminal on MOS transistors of type (model) NSS; two gate terminals of the same kind of transistor and four drain/source terminals on MOS transistors of type (model) PSS, and, finally, to four gate terminals of the same kind.

Resolving a Fragmented Class

If your design uses capacitors, resistors, or inductors, make sure that you treat them consistently either as polarized or nonpolarized devices. If the polarization of resistors is important, make sure you check the **Consider Resistors as Polarized Elements** option on the “Setup—Device Parameters” (page 980) tab in the setup window. If this option is not enabled, some devices may have the positive and negative terminals switched, causing fragmentation.

If your design is fully digital and you have not avoided permuted inputs to gates during layout and schematic design, fragmentation will result. LVS compares elements at a transistor level, and cannot interpret the design at the gate level. Permuted inputs to gates are a common cause of unnecessary fragmentation in digital designs. The **Replace Series MOSFETs** option on the “Setup—Device Parameters” (page 980) tab in the setup window triggers the replacement of series-chain transistors,
such as the series of three $n$-transistors in a three-input NAND gate. In this case, the three $n$-transistors are replaced during the iteration process with an imaginary part with three permutable input terminals, representing the A, B, and C terminals of the gate. Since the transistors are now replaced, any fragmentation caused by permuted inputs to gates is eliminated. If your design contains both digital and analog parts, the digital parts need to be separated and compared separately in order to use this function. Otherwise, the replacement routine also transforms series chain transistors in the analog part of your design.

If your circuit files represent a typical ASIC design situation, you may have represented many parallel MOSFETs in your layout with a single MOSFET in your schematic, with an equivalent width equal to the sum of the widths of your layout MOSFETs. You may also have represented single capacitors and resistors in your schematic with multiple devices in your layout. If either of these cases are applicable, it is worth enabling the **Optimize Network** option on the “Setup—Device Parameters” (page 980) tab in the setup window to optimize the networks before the iteration process begins.

Following is an example of the output generated by LVS for `ex4.vdb`. The output demonstrates the report of nodes in the Fragmentation section.

```
Report of nodes:
****************
Fragmented class of nodes
Ex1_2.spc16 connected to 1 M_NSS_D/S 2 M_NSS_G
  4 M_PSS_D/S 4 M_PSS_G
----------------------------------------------------------
Fragmented class of nodes
Ex1_2.spc3 connected to 3 M_NSS_D/S 1 M_NSS_G
  3 M_PSS_D/S 1 M_PSS_G
Ex1_2.spc15 connected to 3 M_NSS_D/S 1 M_NSS_G
  3 M_PSS_D/S 1 M_PSS_G
----------------------------------------------------------
Fragmented class of nodes
Ex1_2.spc7 connected to 6 M_NSS_G 6 M_PSS_G
----------------------------------------------------------
Fragmented class of nodes
Ex1_2.spc5 connected to 1 M_NSS_D/S 2 M_NSS_G
  2 M_PSS_D/S 4 M_PSS_G
----------------------------------------------------------
Fragmented class of nodes
Ex1_2.spc4 connected to 4 C_POS/NEG 24 M_PSS_BULK
  14 M_PSS_D/S
----------------------------------------------------------
Fragmented class of nodes
Ex1_1.spc5 connected to 1 M_NSS_D/S 2 M_NSS_G
  5 M_PSS_D/S 4 M_PSS_G
----------------------------------------------------------
Fragmented class of nodes
Ex1_1.spcN25(XSUBA/X33/) connected to 3 M_NSS_D/S
  1 M_NSS_G 4 M_PSS_D/S 1 M_PSS_G
Ex1_1.spcN25(XSUBA/X34/) connected to 3 M_NSS_D/S
  1 M_NSS_G 4 M_PSS_D/S 1 M_PSS_G
----------------------------------------------------------
Fragmented class of nodes
Ex1_1.spc1 connected to 6 M_NSS_G 8 M_PSS_G
----------------------------------------------------------
Fragmented class of nodes
Ex1_1.spc8 connected to 1 M_NSS_D/S 2 M_NSS_G
  3 M_PSS_D/S 4 M_PSS_G
----------------------------------------------------------
Fragmented class of nodes
```
From the output we know that there is a discrepancy in the number of elements between the two files. File *ex1_1.spc* has two more *M_PSS* elements than *ex1_2.spc*. This fact is reflected in the following two classes:

---
Fragmented class of nodes
Ex1_1.spc VDD connected to 4 C_POS/NEG
26 M_PSS_BULK 14 M_PSS_D/S
---
Fragmented class of nodes
Ex1_2.spc 4 connected to 4 C_POS/NEG
24 M_PSS_BULK 14 M_PSS_D/S
---

It is reasonable at this point to assume that node 4 and node VDD are supposed to be matched.

The key to finding the difference between the two files lies in concentrating on the differences in the connectivity between the nodes in the fragmented classes. Concentrating on two particular classes, one from each file, will quickly lead to the root of the problem:

---
Fragmented class of nodes
Ex1_2.spc15 connected to 3 M_NSS_D/S 1 M_NSS_G
3 M_PSS_D/S 1 M_PSS_G
Ex1_2.spc3 connected to 3 M_NSS_D/S 1 M_NSS_G
3 M_PSS_D/S 1 M_PSS_G
---

and

---
Fragmented class of nodes
Ex1_1.spcN25(XSUBA/X34) connected to 3 M_NSS_D/S 1 M_NSS_D/S 1 M_NSS_G
M_PSS_G
Ex1_1.spcN25(XSUBA/X33) connected to 3 M_NSS_D/S 1 M_NSS_D/S 1 M_NSS_G
M_PSS_G
---

Node N25 in subcircuit *XSUBA* in file *ex1_1.spc* in two cases has one more connection to a drain-source terminal of a p-transistor than the potential match in the second file, *ex1_2.spc*. Looking at the nodes in their respective netlist files (skipping all but drain-source connections to p-transistors) gives:

*ex1_1.spc:*
M1   N39 N23 N25 VDD PSS L=2 W=5 M=2
M11  2   CLK N25 VDD PSS L=2 W=5 M=2
M4   N39 N25 N23 VDD PSS L=2 W=5 M=1
We are looking at only half the cases. From the second file, **ex1_2.spc**, we are only concentrating on node 3. Similar output is generated for node 15. Looking back at the fragmented classes, we see that node 7 of M11, a potential candidate for an error, has six connections to p-transistors. The potential equivalent node in file **ex1_1.spc**, node 1 (node 1 gets mapped into the CLK node through the hierarchy), has eight connections to p-transistors. Therefore, it is safe to assume that either M11 of file **ex1_2.spc** needs to be duplicated along with the equivalent transistor for the node 15 case, M3411, or that the multiplier M=2 of M11 in file **ex1_1.spc** has to be replaced with M=1. Doing one or the other and rerunning LVS will get through the iteration process to the point of automorphism. (See “Automorph Class Example” on page 1011.)

Often there is more than one problem in a fragmented class. Since fragmentation problems tend to hide each other, the fastest approach is to fix the identified problem and then rerun LVS. In the previous example, only one problem caused the fragmentation.

### Using Device Parameters to Resolve Fragmented Classes

When LVS reports fragmented classes, it will group together elements or nodes that are connected to similarly sized devices. This grouping is often helpful in focusing attention to differences between the two netlists. For example, in the following large netlist, two nodes were inadvertently shorted into a single node. The output report contains:

<table>
<thead>
<tr>
<th>Fragmented class of nodes</th>
<th>cs205.spc: 9 nodes</th>
<th>cs205.cir: 8 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs205.spc 5363</td>
<td>connected to</td>
<td>2 M_ne2_D/S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 M_pe2_D/S</td>
</tr>
<tr>
<td>cs205.cir 418</td>
<td>connected to</td>
<td>2 M_ne2_D/S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 M_pe2_D/S</td>
</tr>
</tbody>
</table>

Node 5363 in cs205.spc was singled out, because the devices that it connected to had parameters that were different from those of the other nodes in this fragmentation class. Of course, if device parameters...
are ignored, the result will be to group all these nodes into a single, undistinguished, fragmentation class (this result is equivalent to the reporting algorithm of LVS V9 and earlier).

**Element Description File Example**

This example illustrates the output generated when an element description file is used during the run. An *element description file* is a text file that describes specialized elements used in the design not identified by LVS. The element description file in this example describes the specialized elements (combs, springs, and plates) used in a resonator for a Micro Electro Mechanical System (MEMS) design. LVS compares the elements down to the level of information provided in the element description file. (For information on the format of this file, see “Element Description Files” on page 972.)

Following are the contents of the element description file *resonator.elm*.

```
|| d comb 1 2 3 4;
|| d plate4 1 2 3 4 5 6 7 8;
|| d fspring 1 2 3 4;
```

Following is an example of the output file *resonator.out*, generated by LVS for *resonator.vdb*. Notice that LVS reports many of these elements as connected to only one pin. Catching nodes with only one connection (floating pins) is a very important check to perform, because not connecting a pin (for example, a bulk node to the substrate) can be fatal in a design. LVS identifies such problems before they become costly.

Reading element definitions from file resonator.elm...
Parsing file layout_resonator.spc...
Flattening network...
Not resolving subckt fspring
Not resolving subckt comb
Not resolving subckt fspring
Not resolving subckt comb
Not resolving subckt plate4
Parsing file schematic_resonator.spc...
Flattening network...

<table>
<thead>
<tr>
<th>Device</th>
<th>layout_resonator.spc</th>
<th>schematic_resonator.spc</th>
</tr>
</thead>
<tbody>
<tr>
<td>fspring</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>plate4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>comb</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Total elements</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Total nodes</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Single-pin nodes</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Nodes in file layout_resonator.spc connected to only one pin
********************
4_e
...
20_m
********************

Nodes in file schematic_resonator.spc connected to only one pin
********************
N1
...
N5
*************** ITERATING ***********************

Iterating...
5% done
10% done
...
100% done

*************** REPORTING AUTOMORPHISM ***********************

Report of elements:
**************

Automorph class of elements
layout resonator.spc XPlateInst_plate4
    fanout: 1=1 2=1 3=1 4=1 5=1 6=1 7=1 8=1
schematic_resonator.spc XMass5_plate4
    fanout: 1=1 2=1 3=1 4=1 5=1 6=1 7=1 8=1

---------------------------------------------------

*************** ITERATION SUMMARY ***********************

4 perfectly matched element class(es)
1 automorphed element class(es)
16 perfectly matched node class(es)

*************** POST ITERATION MATCHING ***********************

Doing detailed trial matching... Step 1 (Match by parameters)

*************** ITERATING ***********************

Doing detailed trial matching... Step 2 (Random matches)
Matched Elements XPlateInst_plate4 and XMass5_plate4

*************** FINAL RESULT ***********************

Note: Networks only compared as far down as specified in the element description file.
Circuits are equal.
30 Netlist Comparison

In netlist comparison, we compare two netlists to verify their supposed equivalence. The two netlists are usually from different sources (such as an S-Edit schematic and an L-Edit layout), but they can also be from two schematics or two layouts; generated by different design editors but represented in the same netlist format; or two revisions of the same schematic.

In a verification run, LVS first reads in the two netlists and compiles a list of elements and a list of nodes. It then uses an iterative process to repeatedly divide classes of elements and nodes into smaller and smaller classes until each element and node can be uniquely identified and compared.

Flattened Netlists

Before comparing circuits, LVS performs pre-processing on each netlist to flatten hierarchical structures and simplify the content. Each file includes only the information that LVS will use to compare circuit descriptions. Commands, comments, and ignored devices or parameters are omitted, and parallel or series elements are merged as specified in the top section of “Setup—Merge Devices” (page 982). The files generated by this processing are called flattened netlists; you can save these files by specifying filenames for Flattened Layout Netlist and Flattened Schematic Netlist in “Setup—Output” (page 978).

Note: If Merge series MOSFETs is selected on the Merge Devices tab, the flattened netlist will not reflect collapsed series MOSFETs. To maintain SPICE compatibility, LVS writes the flattened netlist before merging series MOSFETs.

Multiplicity Parameters

LVS supports the multiplicity parameter M in device statements to indicate multiple parallel devices. When a device statement includes a multiplicity parameter with M > 1, LVS calculates new device parameters to eliminate the multiplicity factor. For a device with M=n, this is equivalent to merging n parallel devices. LVS always replaces devices with M>1 with a single, merged device. This step is not affected by settings on the Merge Devices tab.

For example, the following MOSFET statement indicates three identical devices in parallel:

\[ \text{M1 a b c d NMOS M=3 L=5 W=10} \]

LVS replaces this statement with a single equivalent device without multiplicity:

\[ \text{M1 a b c d NMOS L=5 W=30} \]
Netlist Comparison Basics

LVS compares elements and connections for similar characteristics, not for functionality or purpose. Therefore, it is important that the netlists being compared have the same types of basic circuit elements. If you compared a netlist containing Boolean logic gates and a netlist containing transistors, for example, LVS would find them unequal because it does not construct Boolean logic gates from transistors.

LVS processes subcircuits by flattening them, then individually comparing their constituent elements and nodes. If the two designs resolve to the same hierarchical levels, however, you can explicitly define higher-level subcircuits as elements for comparison in an element description file. For further information on this technique, see “Element Description File Example” on page 1020.

LVS begins by reading in the two netlists and compiling a list of elements and a list of nodes. (An element is any type of logic or circuit component, such as a transistor, resistor, or capacitor. A node indicates a connection: a wire and anything directly attached to it.)

LVS then sorts the elements and nodes into classes. A class is a set of elements or nodes with something in common. For example, LVS might begin by separating the elements into different classes: a transistor class, a resistor class, and so on. Further divisions might divide the transistor class into P transistor and N transistor classes.

Nodes are separated in a similar manner. They might be separated into classes according to the number of elements attached to them. Further separation might be based on the types of elements attached to the nodes.

The process of separating elements and nodes according to topological information is called topological matching. Topological matching groups elements and nodes by types and connectivity, rather than by capacitance or element size, which are not used in the default matching process. Topological matching continues until no further fracturing is possible. Ideally, each class will contain only two members at this point, one from each netlist file. If this state is achieved, the netlists are said to be topologically equal.

If topological equality is not achieved, there can be several explanations. For further information on resolving topological inequality, please see:

- “Fragmented Classes” on page 1023
- “Automorph Classes” on page 1024
- “Permuted Classes in Digital Designs” on page 1026

Fragmented Classes

After topological matching, any class remaining with a different number of members between the two files is called a fragmented class. Fragmented classes are almost always the result of differences between the two netlists, indicating a design error that must be resolved.

LVS cannot resolve fragmented classes because there is not a one-to-one match between elements or nodes in the netlists. If a fragmented class occurs, you should examine the source of the netlist files to determine and resolve the problem.
Resolving Fragmented Classes

When iteration produces fragmented classes, you must examine the fragmented class members and trace them back to their origins in the netlist sources.

If you run LVS as a stand-alone application, you must trace a class member back to its origin using the netlist. This task is much simpler if the element or node has a meaningful name or label in the source design. You will find it helpful to generously label the design with unique, readable names that can be extracted to the netlist.

Understanding LVS output can also help you locate an element or node. During the comparison, hierarchy information is appended onto elements and nodes. For example, an element named \texttt{M7(X3/X2)} refers to a transistor element named \texttt{M7}, which resides in the subcircuit \texttt{X2}, which is instanced from the subcircuit \texttt{X3}. See the “LVS Output Tutorial” on page 1010 for more information about the LVS output format.

If you cannot identify an element or node from its LVS-generated name, try locating the element or node in the netlist file, which may contain additional information. For example, a netlist generated with the option \texttt{Write device coordinates} (in \texttt{Tools > Extract—Output}) might contain a line such as:

\begin{verbatim}
R0 1266 1269 259.6
* R0 Plus Minus ( L B R T ) A = 9.744e2, w= 2.4
\end{verbatim}

The letters \texttt{L}, \texttt{B}, \texttt{R}, and \texttt{T} in the comment line represent four numbers in the netlist file. These numbers would indicate the left, bottom, right and top boundaries of the element recognition layer, respectively. You can look at these coordinates on your layout to find a specific element.

You can also use the option \texttt{Label all devices} (in \texttt{Tools > Extract—Output}) to label all the devices on the layout with ports, where the port text is the device name. Using \texttt{Edit > Find}, you would then be able to find a specific device in the layout.

If the fragmented class is a node, you can count the number of pins on the nodes. This will tell you how many elements are attached to the node.

Another way to identify fragmentation problems is to compare the fragmented classes. If one netlist produces two fragmented element classes with a fanout of one, and the other netlist file produces one fragmented element class with a fanout of two, these three classes may represent the same element (with one pin left without a connection). In such a case, LVS can identify floating pins if you select the option \texttt{Detailed processing information} in “\texttt{Setup—Output}” (page 978).

Automorph Classes

After topological matching, any unresolved class containing an even number of members, half from one netlist and half from the other, is called an automorph class. Automorph classes are not necessarily caused by design errors—they also occur when LVS does not have enough information to distinguish between members of a class. You should resolve automorph classes, however, to confirm the correspondence of the two netlists.

In some cases, members of an automorph class actually do match each other, such as when a class contains identical elements connected in parallel. Because LVS cannot distinguish such elements using its default iteration procedure, it may be unable to resolve them during its initial verification run. For example, LVS would be unable to distinguish two equivalent resistors connected in parallel. In practice, however, such identification would be unnecessary, because the resistors are identical in all respects.
Resolving Automorph Classes

The following sections describe three methods for resolving automorph classes:

- “Preiteration Matching,” below
- “Detailed Trial Matching,” below
- “Parameter Matching” on page 1026

Preiteration Matching

In some cases, you can prevent the formation of an automorph class by providing LVS with enough information to distinguish between the members of a class. This technique is called **preiteration matching**, because LVS performs a preliminary match of specified elements or nodes before its usual iteration.

LVS performs preiteration matching when you instruct it to use a **prematch file**, which contains a list of statements that define equivalent elements or nodes. You enter these statements in a prematch file when you know that a given pair of members are, in fact, identical. The exact format of this file is described in “Prematch Files” on page 972.

To specify the use of a prematch file, check the **Prematch file** option in “Setup—Input” (page 977) and provide the filename and path in the adjacent field.

**Note:** Preiteration matching can significantly increase verification speed, especially for large designs, but it can also prevent LVS from detecting design errors if the prematch file contains erroneous matches. Use a prematch file only when you have previously verified the existence of an automorph class and only when you are certain that the specified elements or nodes are equivalent.

Detailed Trial Matching

Another means of resolving an automorph class is to instruct LVS to run **detailed trial matching**. In detailed trial matching, LVS attempts to resolve automorph classes by making a “guess” match between two class members, then resuming the iteration from that point. The program begins by matching an automorph element pair and then iterates on the automorph node classes until no more iteration can be done. It then matches a pair of automorph nodes and iterates on the remaining automorph elements. LVS continues in this way, alternating between element and node classes until it can go no further.

Detailed trial matching sometimes fractures an automorph class into a fragmented class, but this result would indicate that LVS made an incorrect trial match. In such a case, you should try one or more of the following solutions to resolve the automorph class:

- Rerun the verification with detailed trial matching enabled. LVS temporarily stores a record of its matches in memory, but it does not save them with the VDB file. Therefore, you must make the second attempt at detailed trial matching without closing the VDB file.
- Examine the output file to discover what matching assignments LVS made. With your knowledge of the design, you may be able to make better assignments than those made by LVS and enter them into a prematch file.
- Rerun the verification with parameter matching enabled—see “Parameter Matching,” below.

To specify detailed trial matching, check the option **Automatically perform detailed trial matching to resolve automorph classes** in “Setup—Performance” (page 988). In the event of an automorph class,
LVS will automatically proceed to detailed trial matching. If you do not check this option before the verification run, LVS will prompt you for permission to perform detailed trial matching.

If you run the iteration in batch mode or as part of a verification queue, LVS automatically performs detailed trial matching on all automorph classes whether you check the option or not.

---

**Note:**

LVS stores element and node information internally, and the program’s matching assignments for a particular automorph class depend on the size and location of memory blocks available at a particular time. Therefore, results from detailed trial matching may vary with each run.

---

**Parameter Matching**

The third method of resolving an automorph class is called *parameter matching*. In this method, LVS considers additional user-specified parameters such as capacitance and element size to further distinguish class members that would otherwise form an automorph class. For example, it may be possible to divert automorph nodes with different capacitances into different classes, possibly resolving the automorphism. Such a step may also convert an automorph class to a fragmented one, but you can then take appropriate steps to correct the error that produces the fragmented class.

To specify parameter matching, select the particular parameters you want LVS to consider in the dialog **Setup—Device Parameters**.

---

**Permuted Classes in Digital Designs**

If you select the option **Merge Series MOSFETS**, LVS can catch and identify permuted classes. A permuted class occurs in digital designs when many digital circuits provide terminals that are functionally equivalent, but in a different order in schematic and layout designs. In such a case, LVS will create fragmented classes unless it considers permuted classes.

---

**Warning:**

Permuted classes in an analog design could generate problems and should be avoided.

---

The following illustration demonstrates a permuted class. The two input NOR gate designs shown below are functionally identical, but in topological iteration with series replacement, LVS would note the pin permutation and report an error. The iteration invoked without series replacement would generate four fragmented element classes, each with one element in the class.
Avoiding Permutated Classes

The best way to avoid permuted classes is to standardize pin name assignments in both schematic and layout design. Always assign A to the top pin or the pin closest to $V_{dd}$, for example, and B to the lower pin or the pin closest to $GND$. Standardized pin name assignments such as these will prevent LVS from generating permuted classes.

LVS Algorithms and Limitations

LVS uses an algorithm that repeatedly fractures the classes in an effort to generate a unique classification of the elements and nodes in a netlist. This methodology has several important advantages, but it also has a few limitations.

During the topological matching process, LVS continues to fragment elements and nodes into increasingly smaller and more identifiable classes. In an ideal comparison, the iteration process ends when each class has exactly two members whose features are known. The following diagrams illustrate this procedure.
Notice that bulk nodes are not considered, and the two diagrams of the internal data structures are not identical, indicating fragmented classes if LVS is run without series replacement.

There are several advantages to representing netlists in this way, including:

- Isomorphism can easily be determined since each representation is unique for each circuit.
- No knowledge of driving elements is required.
- No knowledge of the elements and their pins is required.
- There is easy recognition of shorts and opens.

One limitation to this type of netlist representation is that a highly symmetrical design will fail to converge to fragmented classes, as in the diagram below. Exhaustive trial matching of all elements in the second equivalent class yields no self-consistent partitions, and the data structures are not isomorphic. The result, however, is correct.

Resolving Discrepancies

Resolving netlist comparison discrepancies can be a challenging process if the involved circuits are large, because a single problem might create many automorph and/or fragmented classes. Here are a few suggestions for resolving netlist comparison problems, which are applicable to the resolution of both automorph and fragmented classes.
- Liberally label the sources for your netlists. If you do not provide a name for a node or element, LVS constructs one automatically, but the resulting name may be just a number, perhaps concatenated onto another string. Even an LVS-generated identifier can be helpful, however—if you recognize only a portion of a node or element name, you may still be able to identify the problem.

- If possible, recompare the netlists often. Sometimes identifying a single matching element or node will provide LVS with enough information to complete the job. In other cases, a single problem will create a very large fragmented class. Identifying one or two matching pairs from this class and entering them into a prematch file will usually be sufficient for LVS to fracture the large fragmented class into several smaller ones, thus giving you an easier task to perform.
Running LVS from the Command Prompt

You can invoke LVS from a DOS command prompt, specifying input and output files plus all verification options on the , and achieve the same results as you would when running LVS under Windows. Additionally, for batch-mode operation, LVS must be invoked from a DOS command prompt.

To invoke LVS, change to the LVS directory on your hard disk, or make sure that the LVS directory is in your path. Then enter the LVS command at the DOS prompt:

```
LVS [options] "netlist1" "netlist2"
```

`netlist1` and `netlist2` are the two netlist files to be compared. Specify the full path for any file not in the current directory and enclose the entire filename and path in quotation marks.

Filenames and paths with spaces in them are legal, but depending on the DOS operating system version you are using, arguments longer than 112 characters may not be allowed. Exceeding this length will result in the error message “Unrecognized line argument” and termination of LVS processing.

`options` are command-line arguments, which provide additional operating instructions to LVS.

Each of these arguments is discussed in “Options” on page 1031.

Batch-File Syntax

An LVS batch file will contain text similar to the following:

```
C:\Documents and Settings\username\My Documents\Tanner Tools vxx.yy\L-Edit and LVS\LVS\Ex3>start /w ../../../lvs ex3_1.spc ex3_2.spc -p ex3_1.pre -o ex3_2.lvs -nrcl -vfar-fafr
```

Each verification listed in a batch file must begin with the command `start /w`. This command instructs DOS to start LVS with the first set of options and wait for the program to exit before launching it again. Without this command, only the first verification run listed in the batch file would succeed. LVS would fail to run subsequent verifications.

Note that in contrast to the simple command-line example on the previous page, which lists options before the netlist files, this batch-file example lists options following the netlist files. The command `File > Export Batch File` uses this syntax in creating batch files, but in fact, either syntax is legal. If you create a batch file directly in a text editor, you can list command-line options before or after the netlist files. LVS accepts either syntax.

Running a Batch File

To run a batch file, enter the following command from a DOS command prompt:
filename.bat

where filename.bat is the name of the batch file.

LVS will start up and run the verifications listed in the batch file. The program will create a user-specified output file, as specified by the -o command-line option.

You can view the resulting output file using the LVS text window or any other text editor.

**Note:**
LVS does not support launching of multiple instances. If you are running LVS under Windows, you must first exit the program before running a batch file from the DOS.

---

### Options

Options are preceded by a dash or minus sign (-) and are separated by spaces. You can specify options before or after the two netlist files—LVS accepts either syntax.

Where options have arguments, such as -cnnn, the option can be typed with or without a space between the option and the argument—as -cnnn or -c nnn. LVS accepts either syntax.

Except where specifically noted, command-line options are case-insensitive.

#### Ignore Bulk Nodes (-b)

This option is the inverse of Consider bulk nodes (substrate) during iteration matching in the setup window.

The -b option instructs LVS to ignore bulk (substrate) nodes on semiconductor devices during the iteration matching process. (By default, LVS takes bulk nodes into consideration.) A typical digital design will always have the bulk nodes of elements connected to power or ground. Ignoring the bulk nodes while processing such files will reduce memory usage and increase processing speed.

The bulk node parameter is optional in the netlist input file. If you instruct LVS to consider bulk nodes but all of the bulk nodes are not present in the netlists, the iteration process will result in fragmentation.

#### Consider Parameters (-cnnnn)

The -cnnnn option tells LVS what parametric information to consider during iteration matching and trial matching. nnnn is a series of single-digit integers from the following list. This option corresponds to the following options in the setup window (see “Setup—Device Parameters” (page 980).

2. Consider L and W for MOSFETs.
3. Consider AS, AD, PD and PS for MOSFETs.
4. Consider areas of B, D, J, and Q elements.
5. Consider Z0 for T elements.
For example, to consider resistance, capacitance and inductance; L and W; and NRD, NRS, NRG, and NRB, use `-c127`.

**Maximum Value Difference (-dv n)**

This option corresponds to *Maximum element-value tolerance* in the setup window (see “Setup—Device Parameters” (page 980)).

The `-dv n` option, where `n` is a percent value between 0 and 100, defines the maximum amount two parameter values may differ and still compare as equal. This value is often referred to as the *slew rate*. The number is expressed as a percentage of the larger parametric value. The default is 5% (-dv5). Noninteger values are permitted (for example, `-dv4.5`). This option takes effect only when the `-c127` option is used.

For example, if `element1` had a capacitance of 15.5 fF and `element2` had a capacitance of 16.1 fF, they would be considered equivalent with `-dv4`, but not with `-dv3`.

See also “Consider Parameters (-cnnnn)” on page 1031.

**Maximum Geometrical Difference (-dg n)**

This option corresponds to *Maximum geometric-value tolerance* in the setup window (see “Setup—Device Parameters” (page 980)).

The `-dg n` option works in exactly the same manner as `-dv n`, except that it operates on geometrical comparisons, such as L and W for MOSFETs. The default value is 5% (-dg5). This option has an effect only when the `-c3456` option is used.

See also “Consider Parameters (-cnnnn)” on page 1031.

**Element Description File (-e "file")**

This option corresponds to *Element description file* in the setup window (see “Setup—Input” (page 977)).

The `-e "file"` option specifies the location of the element description file named `file`. Specify the full path if needed, and enclose the entire filename and path in quotation marks. LVS recognizes standard SPICE elements such as resistors, inductors, capacitors. If you use a nonstandard element, however, you will need to define it in an element description file and use this option to instruct LVS to read in the file. The netlist files can include elements described in the element description file by using them as a subcircuit element.

**Output File Display Options (-f[fapr])**

These options control the amount of information included in the output file. They correspond to the *Display Options: Output file* checkboxes in the setup window; see “Setup—Output” (page 978).
The `-f[appr]` option instructs LVS to write processing information to the output file. Fragmented classes, permuted classes, automorph classes, and detailed processing information can be saved using this option. The flags `f`, `a`, `p`, and `r` may be included in any combination, as follows.

- `f` Show fragmented classes.
- `a` Show automorph classes.
- `p` Show permuted classes.
- `r` Show detailed processing information:
  - Shows single-connection nodes.
  - If a prematch file is used (-p option), writes the prematched elements to the output file, as well as those elements that the program attempts to postmatch. This is useful for troubleshooting netlists returned as not identical due to fragmentation after automorphism or permutability.
  - If an element description file is used (-e option), lists subcircuits that will not be flattened (i.e., those designated as special elements).
  - Shows a summary of merged series or parallel devices. Logs deletions of shorted and disconnected devices.
  - Lists parasitic devices that were removed or shorted.

### Granularity (-%g=n)

The `-%g=n` option controls the granularity of the percentages displayed while LVS is iterating. The default is `g = 5`; LVS will report its progress to the screen in increments of 5%, based on the total number of nodes and how many it has already processed. Setting `g = 0` turns off the reporting.

**Note:** This option can only be set on the and will only affect the completion percentage reported to the output file. Because LVS does not display the verification window when run in batch mode, use of this option is inappropriate for that mode.

### Flattened Schematic Netlist (-h "file")

This option corresponds to Flattened Schematic Netlist option in “Setup—Output” (page 978). The `-h "file"` option writes a flattened version of the schematic netlist to a separate file named `file`. The flattened netlist includes only the information that LVS will use to compare circuit descriptions. Commands, comments, and ignored devices or parameters are omitted, and parallel or series elements are merged as specified in the top portion of “Setup—Merge Devices” (page 982).

**Note:** If Merge series MOSFETs is selected on the Merge Devices tab, the flattened netlist will not reflect collapsed series MOSFETs. To maintain SPICE compatibility, LVS writes the flattened netlist before merging series MOSFETs.
Fast Iteration (-i)

This option corresponds to **Fast iteration: consider fanout only** in the setup window; see “**Setup—Performance**” (page 988).

The -i option instructs LVS to perform a fast iteration. By default LVS considers fanout and element types when performing topological iteration, but specifying -i instructs LVS to consider only fanout during this process.

Specifying -i increases the speed of the iteration process preparation, where LVS forms the initial element and node classes. When performing a single verification, LVS would normally prompt the user for permission to continue iteration on detection of a mismatch in the element or node count at this point. In batch mode, however, LVS would always continue iteration even on an element or node count mismatch. Therefore, specifying -i is inappropriate for batch-mode operation.

Delete Disconnected Devices (-k)

This option corresponds to **Delete Disconnected Devices** in the setup window; see “**Setup—Merge Devices**” (page 982).

The -k option instructs LVS to delete disconnected devices, which are defined as follows:

- **MOSFETs**
  
  A MOSFET is considered disconnected if both of the following conditions are true:
  
  - The gate terminal is not connected to any other device, and
  - At least one source/drain terminal is not connected to any other device.

- **Other device types**
  
  All other device types are considered disconnected if at least one terminal is not connected to any other device.

List Elements and Nodes (-l "file")

This option corresponds to **Node and element list** in the setup window (see “**Setup—Input**” (page 977).

The -l option instructs LVS to list the nodes and elements into a file of the specified name. Specify the full path if you wish to create this file in a directory other than the current directory.

**Warning:**

This option will typically create a very large output file if the input circuits are even of moderate complexity.

Merge Devices (-mdevice {ALL | model_list})

This set of options corresponds to the top portion of the **Setup—Merge Devices** dialog; see “**Setup—Merge Devices**” (page 982).

The -m option instructs LVS to merge series or parallel devices of the same model into equivalent single devices. You can specify the models that LVS will consider for merge operations, or you can apply the merge option to all models of the specified device type.
For example,

```
-mr ALL
```

instructs LVS to merge instances of the same resistor model and occur in parallel or in series.

The option:

```
-mm /M_n1//M_p1/
```

instructs LVS to merge parallel `n1` MOSFETs and to merge parallel `p1` MOSFETs. In this case, only MOSFETs of the `n1` or `p1` models can be merged; LVS leaves other MOSFET models in their original configuration.

Each `-m` option corresponds to exactly one device type. To set multiple merge options, you must enter the `-m` option for each relevant device type. For example:

```
-mr ALL -mc /C_cap1/ -md ALL ...
```

Arguments for the `-m` option are defined in the following table:

<table>
<thead>
<tr>
<th>device</th>
<th>Device type for which you are specifying the merge option. Each device type is represented by its letter abbreviation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>resistors (parallel and series)</td>
</tr>
<tr>
<td>l</td>
<td>inductors (parallel and series)</td>
</tr>
<tr>
<td>c</td>
<td>capacitors (parallel and series)</td>
</tr>
<tr>
<td>b</td>
<td>GaAsFETs (parallel)</td>
</tr>
<tr>
<td>d</td>
<td>Diodes (parallel)</td>
</tr>
<tr>
<td>j</td>
<td>JFETs (parallel)</td>
</tr>
<tr>
<td>m</td>
<td>MOSFETs (parallel)</td>
</tr>
<tr>
<td>q</td>
<td>BJTs (parallel)</td>
</tr>
<tr>
<td>z</td>
<td>MESFETs (parallel)</td>
</tr>
</tbody>
</table>

`ALL` | Instructs LVS to merge devices that are instances of the same model when they occur in parallel. For `r`, `l`, and `c` device types, LVS also merges series devices of the same model. |

`model_list` | List of models that can be considered for merge operations. The name of each model must be prefixed with the key letter of the device type, followed by an underscore. Type a slash (`/`) at the beginning and end of each model name. For example, |

```
-md /d_d1//d_d2//d_d3/
```

specifies diode models `d1`, `d2`, and `d3`.

### Merging Nonpolarized Devices

LVS can treat `r`, `l`, and `c` devices as either polarized or nonpolarized elements, as specified by the `-n` option on the . If `-n` (nonpolarized treatment of `r`, `l`, and `c` elements) is specified, a circuit may contain devices that can be merged in more than one way. Consider the following example:
C1 A B C=30pF
C2 B A C=30pF
R1 A GND 12k

It is assumed here that even if this is part of a much larger circuit node, node B has only two pins. Merging the capacitors in series, we can replace C1 and C2 with a single capacitor C3 with an equivalent capacitance of 15 pF:

C3 A A C=15 pF
R1 A GND 12k

Alternately, the two capacitors C1 and C2 could be considered to be in parallel, since nodes A and B are interchangeable on a nonpolar device. In this case, merging the two parallel capacitors gives:

C3 A B C=60pF
R1 A GND 12k

Both of these are equally valid ways to merge devices. With the -n option, LVS may use two different approaches to merge devices in the two netlists being compared. This will certainly result in fragmentation, as the two resulting topologies are completely different. If treating elements as nonpolarized will create ambiguous configurations, you should not use the -n option.

**Nonpolarized Elements (-n[rcl])**

These options correspond to the following options in “Setup—Input” (page 977):

- Consider resistors as polarized elements
- Consider capacitors as polarized elements
- Consider inductors as polarized elements

The -n[rcl] option instructs LVS to consider resistor, capacitor, and/or inductor elements as nonpolarized. LVS treats the two terminals of a nonpolarized element as interchangeable during the matching process. If this option is not specified, LVS considers these elements to be polarized; that is, the two terminals of such an element are considered topologically different during the matching process.

The arguments r, c, and l specify which elements LVS is to consider as nonpolarized (r=capacitors, l=inductors, and c=resistors). Specifying -n without an additional argument instructs LVS to consider all these elements as nonpolarized; specifying -nrc1 achieves the same result.

**Note:** In some circuits, use of the -n option may create multiple possibilities for merging devices, leading to fragmentation. These circuits are described in “Merging Nonpolarized Devices” on page 1035.

**Output file (-o "file")**

This option corresponds to the Output file option in “Setup—Output” (page 978). The -o "outfile" option (lowercase o required) creates a separate output file named file.

**Note:** In batch-mode operations, LVS does not write results to the verification window. Therefore, an output file is required to preserve verification results.
**Prematch File (-p "file")**

This option corresponds to **Prematch file** in the setup window (see “**Setup—Input**” (page 977)).

The **-p"file"** option instructs LVS to equate the elements and nodes listed in the prematch file named **file** before beginning the iterative matching process. Specify the full path if this file will not be created in the current directory.

The prematch file is a text file created by the user to equate certain elements and nodes before LVS begins its processing. For further information, see “**Prematch Files**” on page 972.

**Input SPICE Syntax (-pspice, -phspice, -hpspice)**

These options correspond to the SPICE format options for **Layout netlist** and **Schematic netlist** in the setup window; see “**Setup—Input**” (page 977). The default syntax mode for input files is T-Spice/H-Spice. If either input file is in P-Spice syntax, you must specify one of the following options:

- **-pspice**
  - Both input files are in P-Spice format.
- **-phspice**
  - The layout netlist is in P-Spice format, and the schematic netlist is in T-Spice/H-Spice format.
- **-hpspice**
  - The layout netlist is in T-Spice/H-Spice format, and the schematic netlist is in P-Spice format.

**Merge Series MOSFETs (-r {ALL | model_list})**

The **-r** option corresponds to the **Merge series MOSFETs** option, and the **[s]** option corresponds to **Find series MOSFETs that differ in order**. Both options are found in “**Setup—Merge Devices**” (page 982).

The **-r[s]** option allows LVS to replace series chain MOSFETs (of a particular model) with equivalent components, which reduces the processing required. (Instances of two or more different MOSFET models are never merged.) This feature is intended only for fully digital designs, and is not meant for netlists representing analog designs. The inclusion of the **[s]** flag instructs LVS to identify functionally equivalent groups of series MOSFETs that are ordered differently in the two netlists.

**Note:**

Series MOSFETs that differ in order are also called **permuted classes**. See “**Permuted Classes in Digital Designs**” on page 1026 for further discussion.

Arguments for the **-r[s]** option are:

- **ALL**
  - Instructs LVS to merge series chain MOSFETs that are instances of the same model.
**Remove Parasitics (-s test=value)**

This option corresponds to the options in “Setup Window—Parasitics” (page 984). The -s option specifies a criterion by which LVS can identify parasitic devices to remove or short.

Arguments for this option include:

**test**  
Choose one of the following options, which correspond to the checkboxes in “Setup Window—Parasitics” (page 984):

- **rmin** — For every resistor with resistance less than or equal to the specified value, LVS removes the resistor and connects (shorts) the two nodes that were spanned.
- **rmax** — Removes any resistor with resistance greater than or equal to the specified value.
- **cmin** — Removes any capacitor with capacitance less than or equal to the specified value.
- **cmax** — For every capacitor with capacitance greater than or equal to the specified value, LVS removes the capacitor and connects (shorts) the two nodes that were spanned.

**value**  
Maximum or minimum value of the device parameter specified in test.

For example, the options:

```
-srmin=10 -srmax=0.01p
```

instruct LVS to short and remove resistors with resistances less than 10 Ohms, and to remove capacitors with capacitances greater than 0.01 pF. Use a separate -s option to specify each threshold value.

**Flattened Layout Netlist (-t "file")**

This option corresponds to Flattened Layout Netlist option in “Setup—Output” (page 978). The -t "file" option writes the flattened version of the layout netlist to a separate file named file. The flattened netlist includes only the information that LVS will use to compare circuit descriptions. Commands, comments, and ignored devices or parameters are omitted, and parallel or series elements are merged as specified in the top portion of “Setup—Merge Devices” (page 982).
Note: If Merge series MOSFETs is selected on the Merge Devices tab, the flattened netlist will not reflect collapsed series MOSFETs. To maintain SPICE compatibility, LVS writes the flattened netlist before merging series MOSFETs.

Remove Device Models (-u /model1//model2//.../)

This option corresponds to the option Remove device models named: in “Setup Window—Parasitics” (page 984). The -u option instructs LVS to remove all instances of the listed device models from the input netlists.

Each device model name must be prefixed by the key letter for that device type, followed by an underscore. Each device model name must also be enclosed in forward slashes (/). For example,

- u /M_n1//C_cap/

instructs LVS to remove instances of the MOSFET model n1 and of the capacitor model cap.

Screen Display Options(-v[fpar])

These options control the amount of information displayed on the screen. They correspond to the Display Options: Screen checkboxes in the setup window; see “Setup—Output” (page 978).

The -v[fpar] option instructs LVS to display processing information on the screen. Fragmented classes, permuted classes, automorph classes, and detailed processing information can be displayed using this option. The flags f, a, p, and r may be included in any combination, as follows.

f Show fragmented classes.
a Show automorph classes.
p Show permuted classes.
r Show detailed processing information:
  ▪ Displays single-connection nodes.
  ▪ If a prematch file is used (-p option), LVS writes the prematched elements to the screen, as well as those elements that the program attempts to postmatch. This is useful for troubleshooting netlists returned as not identical due to fragmentation after automorphism or permutability.
  ▪ If an element description file is used (-e option), LVS lists subcircuits that will not be flattened (i.e., those designated as special elements).
  ▪ Displays a summary of merged series or parallel devices. Logs deletions of shorted and disconnected devices.
  ▪ Lists parasitic devices that were removed or shorted.
**Note:** These options instruct LVS to display processing information to the verification window. Because LVS does not display the verification window when run in batch mode, use of these options is inappropriate for that mode.

**Delete Shorted Devices (-x)**

This option corresponds to the Delete shorted devices option in “Setup—Merge Devices” (page 982). The -x option instructs LVS to delete shorted devices, in which all device terminals are connected together.

**Yes to All Questions (-y[12])**

The -y[1] option corresponds to Continue on element/node count mismatch, while the -y[2] option corresponds to Detailed trial matching to resolve automorph classes. Both are located in “Setup—Performance” (page 988).

The -y option without any arguments is equivalent to -y12. This instructs LVS to answer “yes” to all program prompts.

**Short Out Device Models (-z /model1//model2/.../)**

This option corresponds to the option Short out device models named: in “Setup Window—Parasitics” (page 984). The -z option instructs LVS to short the terminals of the indicated devices, then remove these devices from the netlist.

Each device model name must be prefixed by the key letter for that device type, followed by an underscore. Each device model name must also be enclosed in forward slashes (/). For example,

```
-z /R_res//C_cap/
```

instructs LVS to short and remove instances of the resistor model res and of the capacitor model cap.
automorph class
A class with an even number of four or more members, half from each netlist, in which there is insufficient information to further resolve the class. The members may be identical, but additional information (such as parameters) may be necessary to differentiate the members.

batch file
A text file containing one or more command-line invocations of LVS and appropriate setup information—input files and verification options—for each verification run.

class
A set of elements or nodes with something in common, such as topological or parametric characteristics. Types of classes include automorph, fragmented, and permuted.

detailed trial matching
A process used by LVS to try to resolve automorph classes. LVS assigns a matching pair of members and continues the iteration process from that match.

element
Any type of logic or circuit component (transistor, resistor, or capacitor).

element description file
A text file that describes specialized or custom elements which are used in the design but not recognized by LVS.

fragmented class
A class with a different number of members from each netlist. This type of class is usually the result of one or more design errors.

netlist
A textual description of the connectivity of a design.

node
An electrical connection between one or more ports, labels, or wires.

node and element list
A text file that contains lists of all the matching and unresolved nodes and elements in the two netlists being compared. The lists are broken down by each LVS iteration.

parameters
Information in a netlist about each element and node in addition to topological characteristics. For example, node capacitance and element size.
parameter matching

The process of using parameters in addition to topological characteristics to differentiate members of a class.

permuted class

A class containing series chain MOSFETs whose terminals are functionally equivalent, but in a different order, in the designs being compared. Occurs in digital designs during the optional replacement of series chain MOSFETs with equivalent components.

prematch file

A user-supplied text file that lists the equivalent members of an automorph class.

resolution

Classes are resolved when their members are matched (through detailed trial matching or another process). Occurs when the members of a netlist are equivalently matched to the members of the other netlist.

SPICE

A netlist format commonly used for circuit simulation or comparison.

slew rate

The maximum percentage by which two parameter values may differ and still compare as equal.

topological characteristics

Information in a netlist regarding element type and connectivities of each element and node.

topological matching

The default LVS iteration process that uses topological characteristics to match elements and nodes.

verbosity level

The amount of information displayed in the verification window during a verification run.

verification queue

A dialog that contains a list of consecutive verification runs.

verification run

The set of iterations LVS executes to compare netlists.

verification setup

Information required for an LVS verification run, including input and output files and verification options. The information is entered in the setup dialogs and can be saved in a verification database file.
The L-Edit User-Programmable Interface (UPI) is a powerful set of tools for automating, customizing, and extending the L-Edit commands and functions, adding enormously to its power and flexibility.

UPI uses C/C++ language macros that describe actions or sets of actions to be performed automatically.

UPI scripts are precompiled as dynamically linked libraries (DLLs) in the background and then executed (rather than being interpreted), or they can be explicitly saved as DLLs for later distribution and use. L-Edit example macros (which previously had to be compiled by the user) have been updated to work correctly with the new compiler.

Macros are written using a vast set of UPI functions, variables, and data types that specify and modify the whole range of L-Edit operations. Sets of macros can be loaded together, nested recursively or accessed individually.

The “UPI Functions Reference” (page 1072) chapter has descriptions of all the UPI functions.

Using Macros-The Basics

There are several steps to using UPI macros:

“Writing a Macro” (page 1046)

Write, name and save macros as files that consists of one or more C/C++-language macro routines.

“Binding Macros to Hot Keys” (page 1049) and “Binding Macros to Menu Items” (page 1049)

A macro can be associated with a keyboard shortcut and/or a menu command.

“Creating a Copy-Protected DLL” (page 1065) and “Opening a Copy-Protected DLL” (page 1065)

A macro can be saved as a fully compiled dll.

“Loading a Macro” (page 1068)

Macros are loaded with a special macro interface.

“Running a Macro” (page 1071)

Execute the macro.
Using UPI Macros-The Basics

To create a new UPI cell, use **File > New** and select the **UPI Macro** option under File type. L-Edit will open a file in its text editor that contains the template for a UPI macro.

```c
#include <cstdlib>
#include <cstdarg>
#include <cstdio>
#include <cstring>
#include <cctype>
#include <cmath>

#define EXCLUDE_EDIT_LEGACY_UPI

#include <ldata.h>
/* Begin -- Uncomment this block if you are using L-Comp. */
// #include <loomp.h>
/* End */

extern "C" {
    void Macro1(void);
    int UPI_Entry_Point(void);
}

void Macro1(void)
{
    LCell pCell = LCell_GetVisible();
    LFile pFile = LCell_GetFile(pCell);

    // TODO: Insert your code here
}

int UPI_Entry_Point(void)
{
    LMacro_BindToMenuAndHotKey_v9_30("Tools", NULL /*hotkey*/,
                                    "Macro1", "Macro1", NULL /*hotkey category*/);
    return 1;
}
```
Format for a Compiled (.dll) Macro

In outline form, the code for a UPI macro in DLL has the following structure:

<include files>

<macro function 1>
<macro function 2>
<macro function 3>
.
.
.

<UPI_Entry_Point function>

For example, the complete code for the Active-to-Metal macro in the ...samples\upi directory is as follows:

```c
#define EXCLUDE_LEDIT_LEGACY_UPI

#include <data.h>

/* Begin -- Uncomment this block if you are using L-Comp. */
#include <lcomp.h>
/* End */

extern "C" {
void Macro1(void);
int UPI_Entry_Point(void);
}

// TODO: Insert your code here

int UPI_Entry_Point(void)
{
  LMacro_BindToMenuAndHotKey_v9_30("Tools",
      NULL /*hotkey*/,
      "Macro1", "Macro1", NULL /*hotkey category*/);
  return 1;
}

// TODO: Insert your code here
Enter your UPI functions here.

Use this code to attach a UPI macro to a set of keystrokes and to add it to a menu.

This statement make the C language macros, which are now superseded by C++ functions, unavailable.

This set the default so that if you are using L-Comp, you must delete this block. (Note that this is an inversion of the pre-v15 default.jkljwerqwq

This is a directive to use C style calling conventions so the same functions can be called from C or C++ programs.

extern "C" {
void Macro1(void);
int UPI_Entry_Point(void);
}

This is a directive to use C style calling conventions so the same functions can be called from C or C++ programs.

// TODO: Insert your code here
Enter your UPI functions here.

Use this code to attach a UPI macro to a set of keystrokes and to add it to a menu.

This statement make the C language macros, which are now superseded by C++ functions, unavailable.

This set the default so that if you are using L-Comp, you must delete this block. (Note that this is an inversion of the pre-v15 default.jkljwerqwq

This is a directive to use C style calling conventions so the same functions can be called from C or C++ programs.

extern "C" {
void Macro1(void);
int UPI_Entry_Point(void);
}

This is a directive to use C style calling conventions so the same functions can be called from C or C++ programs.

// TODO: Insert your code here
Enter your UPI functions here.

Use this code to attach a UPI macro to a set of keystrokes and to add it to a menu.

This statement make the C language macros, which are now superseded by C++ functions, unavailable.

This set the default so that if you are using L-Comp, you must delete this block. (Note that this is an inversion of the pre-v15 default.jkljwerqwq

This is a directive to use C style calling conventions so the same functions can be called from C or C++ programs.

extern "C" {
void Macro1(void);
int UPI_Entry_Point(void);
}

This is a directive to use C style calling conventions so the same functions can be called from C or C++ programs.

// TODO: Insert your code here
Enter your UPI functions here.

Use this code to attach a UPI macro to a set of keystrokes and to add it to a menu.

This statement make the C language macros, which are now superseded by C++ functions, unavailable.

This set the default so that if you are using L-Comp, you must delete this block. (Note that this is an inversion of the pre-v15 default.jkljwerqwq

This is a directive to use C style calling conventions so the same functions can be called from C or C++ programs.
Writing a Macro

Example 1: Hello World

Here is an example of the structure and content of a simple `.cpp` UPI application called `Hello, World!`, which creates the message box shown below.

The code for `Hello, World!` is provided in the shipping examples installation directory, typically `...\My Documents\Tanner EDA\Tanner Tools vxx.y\Features By Tool\L-Edit\UPI Macros\HelloWorld`.

You can create and edit a macro in any text editor. However, the L-Edit text editor offers the advantage of syntax highlighting and also error marking (see “Debugging Macros” on page 1048.)

Select File > New and choose Text from the File Type list then click OK to open a new text window.

Format Outline

A compiled macro file consists of two sections:

- A an application module containing macro function definitions.
- A call to the macro registration function, which is defined within the application module.

In general outline form, the code for a compiled UPI application has the following structure:

```cpp
module <modulename> { 
```
<include files>

<macro function 1>
<macro function 2>
<macro function 3>
.
.
.

<macro registration function>
}
<call to macro registration function>

Defining the Macro Module

The first line contains the module name. The module name should be unique so that it will not replace another module loaded at the same time. In hello.cpp, the module name is Hello_World_module.

To access UPI function definitions, the module definition must include the header file ldata.h. Other required header files should also be included at the beginning of the module definition. (For information about header files, see “UPI Include Files” on page 1071.)

The module outline for Hello, World! is as follows:

module Hello_World_module {
#include “ldata.h”

<Macro function to display message box>
<Macro registration function>
}
<Call to macro registration function>

Displaying a Message Box

“LDialog_MsgBox” (page 1077) is a UPI function that displays a message box containing the specified text. In this example, the macro function HelloWorldMacro invokes LDialog_MsgBox to show the string “Hello, World!” in a message box.

void HelloMacro( void )
{
    LDialog_MsgBox ( “Hello, World!” );
}

Registering the Function as a Macro

To complete a module definition, you must register the defined macro function(s) in a macro registration function. The macro registration function for the module Hello_World_module consists of the following code:

void hello_world_macro_register ( void )
{
    LMacro_Register ( “Hello, World!”, “HelloMacro” );
}

The function hello_world_macro_register registers the function HelloWorldMacro as an available macro. “LMacro_Register” (page 1100) registers a macro name that will be displayed in the Macros list.
of the Tools > Macro dialog and associates this name with the specified function. In this example, LMacro_Register associates the name Hello, World! in the Macro dialog with the macro function HelloWorldMacro.

If a module includes multiple user macro functions, the macro registration function should register each of them individually using multiple LMacro_Register function calls.

The complete code for the Hello, World! macro is shown below. Note that the last line of the macro file calls the macro registration function.

```cpp
module Hello_World_module {
    #include "ldata.h"

    void HelloMacro() {
        LDialog_MsgBox ( "Hello, World!" );
    }

    void hello_world_macro_register ( void ){
        LMacro_Register ( "Hello, World!", "HelloMacro" );
    }
}
hello_world_macro_register();
```

Save your code using the .cpp filename extension. Use this extension whenever you name a macro file that you want L-Edit to recognize as a macro.

**Debugging Macros**

In addition to formatting code files with color and structural indentations, if there are errors in your UPI code L-Edit will place a marker in the margin of the text editor on each line that has an error.

The error log shows a description of each error that includes the line number, and will jump to the error line in the text editor when you double-click on the error in the log.
A few common mistakes to avoid:

- Make sure that the interpreter header file location is properly set. You set the header file location using the Setup Application dialog—see “Entering Library References” on page 1070.
- Make sure the C-library functions in your code are present in the header files specified in the Interpreter Setup dialog. If they are not present, L-Edit will not be able to interpret your code.
- Make sure you have proper permissions to the directory where you create your log file.

**Binding Macros to Hot Keys**

You can modify the L-Edit user interface so that you can execute your macro from a hot key (or a menu item, see “Binding Macros to Menu Items,” below) instead of using the Macro dialog. The process is known as binding.

You bind a macro to a hot key using “LMacro_BindToHotKey” (page 1101). This function registers the macro and binds it to the specified hot key. All the allowed key codes are defined in ldata.h. The binding operation in a macro will overwrite any existing key binding. To view existing key bindings for all macros, use “Setup Application–Keyboard” (page 67).

For example, the following code binds the macro “Contact, Active-Metal1” to the F1 key:

```c
LMacro_BindToHotKey ( KEY_F1, "Contact, Active-Metal1", 
                      "Contact_Active_Metal1" );
```

**Binding Macros to Menu Items**

This section describes how to make your macro accessible as an L-Edit menu command. The UPI function “LMacro_BindToMenu” (page 1102) registers the macro and binds it to a user-specified menu item.

```c
void LMacro_BindToMenu( char *menu, char *macro_desc, void *function);
```

For example, the following command will bind Contact_Active_Metal1 to the Tools menu.

```c
LMacro_BindToMenu ( "Tools", "My Contact, Active-Metal1", 
                    "Contact_Active_Metal1" );
```

**Checking for Cell Locks and Reservations**

Before modifying a cell, a UPI macro should check if the cell is locked. If the macro is to run on an OpenAccess database, the macro should also check if the cell is reserved, and get the reservation if it is not. This can all be done with a call to LCell_GetLock, as shown in the following example.

**Example 2: Draw Box**

```c
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include "ldata.h"

void DrawBox( void )
{
    LFile File;
```
LCell Cell;
LLayer currentLayer;

// Get Cell, File, and current Layer
Cell = LCell_GetVisible();
File = LCell_GetFile(Cell);
currentLayer = LLayer_GetCurrent( File );

// Check lock and get reservation before modifying cell
if (LCell_GetLock(Cell) != 0)
    return;

// Draw a box
LBox_New( Cell, currentLayer, 0, 0, 100, 100 );
LDisplay_Refresh();

int  UPI_Entry_Point( void )
{
    LMacro_BindToHotKey ( KEY_B, "Draw a Box", "DrawBox" );

    return 1;
}

How to compile UPI with Visual Studio 2005

[1] The first step is to set up paths for UPI in Microsoft Visual Studio 2005. Go to Tools > Options and select VC++ Directories under Projects and Solutions.

Change Show directories for: to Include files and press the New Line button. Then press the browse button and browse to the DLL_Include directory of the current installation (typically C:\Program Files\Tanner EDA\Tanner Tools vxx.y\upi\DLL_Include).
Next, change **Show directories for:** to **Library files** and press the **New Line** button. Again, browse to the **Include** directory of the current installation.

Using a NON-MFC UPI macro in Microsoft Visual Studio 2005

1. Use **File > New > Project.**
[2] Select project type **Win32** under **Visual C++**, choose the template **Win32 Project** and type your project name in the **Name** field.

[3] Click **Next**.
Then select **DLL** under **Application type**, and press **Finish**.

Visual Studio will create a `.cpp` file.
Add a line to include the UPI macro function `ldata.h`, and add the `UPI_Entry_Point` function.

Right-click on the **Source Files** entry in the **Solution Explorer** and select **Add > New Item**.
[8] Select **Module-definition file (.def)** and give it a name.

![Adding new item](image1.png)

[9] Add an **EXPORTS** section to the def file with **UPI_Entry_Point** and the name of the UPI macro function.

![Adding exports](image2.png)

[12] Set **Enable Minimal Rebuild** to **Yes (/GM)** and set **Runtime Library** to **Multi-threaded debug (/MTd)** in the **Debug > C/C++ > Code Generation** property page.

![HelloWorld Property Pages](image)


![HelloWorld Property Pages](image)
[14] Change **Character Set** to **Use Multi-Byte Character Set** in the **Release > General** property page.

[15] Change the **Warning Level** to **Level 4 (/W4)** and **Debug Information Format** to **Disabled** in the **Release C/C++ > General** property page.

[16] Set **Enable String Pooling** to **Yes (/GF)**, **Runtime Library** to **Multi-threaded (/MT)**, **Buffer Security Check** to **No (/GS-)**, **Enable Function-Level Linking** to **Yes (/Gy)**, and **Enable Enhanced Instruction**

[17] Set Inline Function Expansion to Any Suitable (/Ob2), Enable Intrinsic Functions to Yes (/Oi), and Favor Size or Speed to Favor Fast Code (/Ot) in the Release > C/C++ > Optimization property page.
Add **MAKE_DLL** and **COMPILE_AS_V2005** to **Preprocessor Definitions** in the **Release > C/C++ > Preprocessor** property page.

**Example 3: Layout Palette**

In this section, you will learn to create a layout palette that always stays on screen. To follow this lesson, you will need some knowledge of the Windows application programming interface (API).

Each button on this layout palette represents a user macro that creates layout geometry such as a MOSFET, spiral, or gear.

The layout palette is called a modeless dialog because it can remain open while you perform other work in L-Edit. You can use the Windows API to associate tool tips and bitmaps with its buttons.

To implement a layout palette, you must write a DLL with a **UPI_Entry_Point()** function that registers the layout palette macro. Running that macro will make the layout palette appear on your screen.

The following procedure explains how to create a DLL that brings up a layout palette.

- Use a resource editor to create resources for the layout palette.
Create a \texttt{UPI\_Entry\_Point()} function that registers the layout palette macro.

Write code for displaying and managing the layout palette.

Write macros that will generate layout for every button in your layout palette—for example, \texttt{gear.cpp}, \texttt{mosfet.cpp}, \texttt{polarary.cpp}, \texttt{spiral.cpp}, or \texttt{spring.cpp}.

Compile the DLL.

L-Edit comes with source code that you can adapt for use with your own layout palette. The directory \texttt{<installdir>\Samples\UPI\dll\palette} contains the complete source code required to create the layout palette DLL.

Creating Resources

In order to display the graphic elements of a layout palette, including bitmaps, push buttons, and a modeless dialog box, you must create these items as \textit{resources} for your DLL project.

Visual C++ provides a resource editor to create and modify project resources. To access the resource editor, select \textit{File} > \textit{New} from the Visual C++ menu. In the \textit{New} dialog, choose the \textit{Files} tab and select \textit{Resource Script} from the list of file types. Assign the script a name in the File Name field and click OK.

To add an existing resource script to your project, right-click on \textit{Source Files} in the \textit{Projects} file tree and select \textit{Add file to folder} from the pop-up menu. In the \textit{Insert Files into Project} dialog, select the desired resource file (\texttt{*.rc}) and click \textit{OK}.

After you have added a resource file to your project, you can, create, import, and edit resources. To add a new resource, change the \textit{Projects} view to \textit{ResourceView} and right-click on the resource script. Select \textit{Insert} to display a list of resource types:

\begin{itemize}
\item Accelerator
\item Bitmap
\item Cursor
\item Dialog
\item HTML
\item Icon
\item Menu
\item String Table
\item Toolbar
\item Version
\end{itemize}

From this dialog you can create a new resource or import an existing resource, such as a bitmap file. To open a resource for editing, simply double-click the resource name in the \textit{Projects} tree.

The resources needed to create the layout palette described in this section are provided in \texttt{<install\_dir>\Samples\UPI\dll\palette\dll.rc}. To load the resources, simply add \texttt{dll.rc} to your project \textit{Source Files}.
UPI_Entry_Point() Function

The following sample code contains the `UPI_Entry_Point()` function that registers the layout palette macro. When executed, the layout palette macro will call `MainFunction()`, which displays the layout palette.

```
#define STRICT
#include "windows.h"
#include "ldata.h"
extern void LWindow_GetParameters(void **hInst, void **hWnd, void **hLib);
extern void MainFunction(HINSTANCE hInst, HWND hWnd, HINSTANCE hLib);

HINSTANCE hInst=NULL;
HWND hWnd=NULL;
HINSTANCE hLib=NULL;

void LayoutPalette ( void )
{
    MainFunction(hInst, hWnd, hLib);
}

int UPI_Entry_Point ( void )
{
    LWindow_GetParameters( (void**)&hInst, (void**)&hWnd, (void**)&hLib);
    LMacro_Register ( "Layout Palette", "LayoutPalette" );
    return 1;
}
```

This code is provided for you in `<install_dir>\Samples\UPI\dll\palette\user.c`.

Displaying and Managing the Palette

Here is sample code for a file called `dll.c`, which is also provided in your samples directory for this example. This code implements `MainFunction`, which displays and manages the layout palette.

You can copy this code and use it to create your own layout palette.

```
BOOL CALLBACK DlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    HANDLE_DLG_MSG( hDlg, WM_COMMAND, DlgOnCommand );
    HANDLE_DLG_MSG( hDlg, WM_INITDIALOG, OnInitDialog );
    }
    return FALSE ;
}

void DlgOnCommand ( HWND hDlg, int iID , HWND hwndCtl , UINT uCodeNotify )
{
    switch( iID )
    {
    case IDC_BUTTON_5:
        MosfetMacro();
        break;
    }
BOOL OnInitDialog ( HWND hDlg, HWND hwndFocus, long lInitParam )
{
    HBITMAP hBmp;
    TOOLINFO info;
    HWND hToolTip;

    hBmp = LoadBitmap(l_hLib, MAKEINTRESOURCE(IDB_BITMAP5));
    SendMessage(hDlg, IDC_BUTTON_5, BM_SETIMAGE, 0, (LPARAM)hBmp);
    FreeResource(hBmp);
    return TRUE;
}

void OnDestroy ( HWND hDlg )
{
    DWORD error;
    if ( hDlg )
    {
        DestroyWindow( hDlg );
        l_hDlg = NULL;
    }
}

void OnClose ( HWND hDlg )
{
    DWORD error;
    DestroyWindow( hDlg );
    l_hDlg = NULL;
}

void MainFunction(HINSTANCE hInst, HWND hWnd, HINSTANCE hLib)
{
    /* Check if the resources have already been loaded */
    if (!loaded) {
        HRSRC hRes;

        hRes = FindResource(hLib, "PaletteDialogBox", RT_DIALOG);
        hResLoad = (HRSRC )LoadResource(hLib, hRes);
        hTmpl = (DLGTEMPLATE *)LockResource(hResLoad);
        hDlg = CreateDialogIndirect(Null, hTmpl, NULL, DlgProc);
        l_hDlg = hDlg;
        UnlockResource(hResLoad);
        FreeResource(hRes);
    }
    loaded = 1;
    hDlg = l_hDlg;
    ShowWindow(hDlg, SW_NORMAL);
}

BOOL WINAPI DllMain( HANDLE hDLL, DWORD dwReason, LPVOID lpReserved )
{
    switch( dwReason ) {
    case DLL_PROCESS_DETACH:
        if ( l_hDlg != NULL ) {
            DestroyWindow(l_hDlg);
            l_hDlg = NULL;
            loaded = 0;
        }
        break;
    }
return TRUE;
}

Macro Definitions

The following sample code creates a gear using the specified parameters. You can copy this code and adapt it for use with your own layout palette. It is provided in your samples directory as gear.c.

```c
void GearMacro ( void )
{
    LPoint    Polygon [ 100 ];
    float    Angle2, Angle3, R2, R3, Tooth_Angle;
    LCoord    R_Inner, R_Outer, Teeth_Count, Teeth_Width, Tooth;
    LCell    Cell_Draw = LCell_GetVisible ( );
    LFile    File_Draw = LCell_GetFile ( Cell_Draw );
    LPoint    Translation = LCursor_GetPosition ( );
    LDialogItem Dialog_Items [ 3 ] = { { "Inner Radius", "175" },
                                      { "Outer Radius", "200" },
                                      { "Teeth Count ", "15 " });

do {
    if ( !LDialog_MultiLineInputBox ( "Gear Properties", Dialog_Items, 3 ))
        return;
    R_Inner   = atol ( Dialog_Items [ 0 ].value );
    R_Outer   = atol ( Dialog_Items [ 1 ].value );
    Teeth_Count = atol ( Dialog_Items [ 2 ].value );
    Teeth_Width = 6.283185307 * R_Inner / ( 2 * Teeth_Count );
} while ( ( Teeth_Count < 3 ) || ( Teeth_Count > 25   )
    || ( Teeth_Width < 2 ) || ( R_Inner   >= R_Outer )
    || ( R_Inner   < 2 ) );

for ( Tooth = 0; Tooth < Teeth_Count; Tooth++ )
{
    Tooth_Angle = 6.283185307 * Tooth / Teeth_Count;
    R2 = sqrt ( R_Outer * R_Outer + Teeth_Width * Teeth_Width / 4.0 );
    R3 = sqrt ( R_Inner * R_Inner + Teeth_Width * Teeth_Width / 4.0 );
    Angle2 = atan2 ( Teeth_Width / 2.0, R_Outer );
    Angle3 = atan2 ( Teeth_Width / 2.0, R_Inner );
    Polygon [ 4 * Tooth + 0 ] = LPoint_Set ( R3 * sin ( Tooth_Angle - Angle3 ) + Translation.x,
                                             R3 * cos ( Tooth_Angle - Angle3 ) + Translation.y );
    Polygon [ 4 * Tooth + 1 ] = LPoint_Set ( R2 * sin ( Tooth_Angle - Angle2 ) + Translation.x,
                                             R2 * cos ( Tooth_Angle - Angle2 ) + Translation.y );
    Polygon [ 4 * Tooth + 2 ] = LPoint_Set ( R2 * sin ( Tooth_Angle + Angle2 ) + Translation.x,
                                             R2 * cos ( Tooth_Angle + Angle2 ) + Translation.y );
    Polygon [ 4 * Tooth + 3 ] = LPoint_Set ( R3 * sin ( Tooth_Angle + Angle3 ) + Translation.x,
                                             R3 * cos ( Tooth_Angle + Angle3 ) + Translation.y );
}
LPolygon_New ( Cell_Draw, LLayer_Find ( File_Draw, "Metal1" ), Polygon, 4 * Teeth_Count);
LCell_MakeVisible ( Cell_Draw );
LCell_HomeView(Cell_Draw);
}
```

Copy-Protecting Macro DLLs

This section describes how to use copy-protected UPI macro DLLs and how to copy-protect your own macro DLLs.
Opening a Copy-Protected DLL

When you load a copy-protected DLL, L-Edit will display a dialog similar to the following:

The following procedure explains how to use a copy-protected DLL.

- Call the DLL vendor and ask for a password.
- Provide the vendor with your L-Edit serial number and request a key for the DLL. The vendor will compute the password and authorize use by providing a function:

  \[
  \text{password} = f(\text{L-Edit Serial number}, \text{Request key for DLL})
  \]

- Enter the password in the Password field. The DLL will compare the password with the internally generated password. If they match, the DLL will be loaded.

  If the password succeeds, it will be stored in the system registry. That way, you will not have to retype the password every time you try to load the DLL.

Creating a Copy-Protected DLL

You can add copy-protection to any DLL by adding a password verification routine to the source code. To generate copy-protection functions, you will need some knowledge of the Windows application programming interface (API).

The following sections describe the steps involved in adding copy-protection to your DLL macro.

Initiating Password Verification

UPI_Entry_Point is the first function called after the DLL initialization routines have been invoked to set up UPI function pointers. It is thus an appropriate place for L-Edit/UPI to check for copy protection.

Use UPI_Entry_Point to perform the following functions:

- Call LWindow_GetParameters to initialize the parameters needed to create Windows dialog boxes from within the DLL.
- Call the Windows API function GetModuleFileName to return the name of the DLL file.
Call a user-defined function named **VerifyDLLPassword** to get and verify a password for the DLL.

Register the macro and optionally bind it to a hot key or menu item.

A sample version of the **UPI_Entry_Point** function is shown below.

```c
HINSTANCE hMyInst; /* Application Instance */
HWND hMyWnd; /* Parent handle */
HINSTANCE hMyLib; /* Handle to the DLL */

int UPI_Entry_Point( void )
{
    char filename_buf[255];

    /* Initialize Graphical Window Parameters */
    LWindow_GetParameters((void **)&hMyInst, (void **)&hMyWnd, (void **)&hMyLib);
    GetModuleFileName( (HMODULE )hMyLib, filename_buf, 255);

    /* Verify password for this DLL */
    if (!VerifyDLLPassword(filename_buf))
        return 0;

    /* Register the macro */
    LMacro_BindToHotKey ( KEY_F2, "Contact, Active-Metal1...", "Contact_Active_Metal");

    return 1;
}
```

**Verifying the Password**

**UPI_Entry_Point** calls the user-defined function **VerifyDLLPassword()** to obtain and verify the user’s password. The basic tasks of **VerifyDLLPassword** are:

- Internally generate a correct password for the DLL. You can use almost any algorithm to generate a password, but it will generally be a function of the DLL filename and a unique identification number, such as the serial number of L-Edit. You can retrieve the serial number of L-Edit using the function **LUpi_GetSerialNumber**.

- Check the registry stored password corresponding to the DLL filename. If one is found, compare this to the internally generated password. If the stored password and the generated password are identical, return SUCCESS.

- If no password is found in the registry or if the registry password is incorrect, display a dialog box to request a password from the user.

- Compare the user-supplied password to the internally generated password. If the user-supplied password and the generated password are identical, store the new password in the registry and return SUCCESS. Otherwise, return FAILURE.

In outline form, the code for the **VerifyDLLPassword** function looks like this:

```c
int VerifyDLLPassword( char *dll_file_name )
/* DESCRIPTION: calculate a challenge based on the dll file and request the password from the user. If password matches with one produced internally, then return 1; else return 0 */
```
/* Extract the filename from the input variable DLL_file_name */
_splitpath(dll_file_name, drive, dir, fname, ext);
strcat(fname, ext);

/* Generate a password from the filename and the L-Edit serial
number */
challenge = f(fname, LUpi_GetSerialNumber());

/* Check the registry for a previously entered password */
password_found_in_registry = get_password_from_registry(fname, reg_pass);

/* If a password is found, check it against the internally generated
password (challenge). If the password fails or if no password is
found, request a password from the user. Otherwise, return
SUCCESS. */
if ( (password_found_in_registry != SUCCESS) ||
( (password_found_in_registry == SUCCESS) &&
( strcmp(reg_pass, challenge) != 0)) )
{
  if (!GetDLLPassword(usr_pass))
    return (FAILURE);
  else if (strcmp(reg_pass, challenge) == 0)
    return SUCCESS;
}

/* Compare the user-supplied password with the internally generated
password (challenge). If they match, store the new password in the
appropriate registry key */
if ( strcmp(usr_pass, challenge) == 0) {
  store_password_in_registry(fname, usr_pass);
  return SUCCESS;
}

/* Otherwise, notify the user that the password is incorrect and
return FAILURE */
else {
  LDialog_AlertBox("Incorrect password!");
  return FAILURE;
}

Additional Support Routines

The outline of VerifyDLLPassword, above, relies on several additional support routines. A summary of the
necessary functions is provided below; refer to your Visual C++ documentation for help constructing these functions.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_password_from_registry( fname, reg_pass )</td>
<td>Checks the registry for a current password definition and retrieves it, if one is found.</td>
</tr>
<tr>
<td>f( fname, LUpi_GetSerialNumber() )</td>
<td>Creates an internally generated password from the DLL filename and the user’s L-Edit serial number.</td>
</tr>
<tr>
<td>store_password_in_registry( fname, usr_pass )</td>
<td>Stores the user-supplied password in the registry key that corresponds to the DLL filename.</td>
</tr>
<tr>
<td>GetDLLPassword( usr_pass )</td>
<td>Displays a dialog to request the user’s password.</td>
</tr>
</tbody>
</table>
Loading a Macro

The **Macro** dialog allows you to load, edit, and compile macros. Choose **Tools > Macro** to open the **Macro** dialog.

### Macros
Lists the macros that are currently loaded. Click on a macro to select it.

### Run
Executes the macro highlighted in the Macros list.

### Close
Closes the **Macro** dialog.

### Macro Files
Shows the name and complete path of all loaded macro files. Click on a macro file to select it.

### Load
Invokes the **Open** dialog so you can load a macro.

### Unload
Unloads the highlighted macro file.

### Edit
Opens the C/C++ file in the L-Edit text editor.

### Setup
Opens the **Setup Application—UPI** dialog where you can specify the interpreter setup parameters, library and include files. See “Entering Library References” on page 1070 and “Setup Application—UPI” on page 71.

### Save as Dll
Automatically compiles and saves the C/C++ language file as a DLL.

### Load Files at Startup
When this box is checked, the macros and macro files that are currently loaded will be automatically loaded when L-Edit is next started.
Clicking **Load** in the **Macro** dialog launches the **Open** dialog. You can load macros in two forms, as interpreted C/C++ code (`.c` or `.cpp`) which L-Edit compiles in background when you run the macro, or as an already compiled dynamically-linked library file (`.dll`).
**Entering Library References**

Before you run a macro, you must first set the path to any header files, libraries or include files the macro calls. To enter these paths, click **Setup** in the **Macro** dialog or use **Setup > Application—UPI**.

**Search path for include files and libraries**
Enter the complete path of directories containing any files or libraries referenced by macros currently loaded into L-Edit. See “UPI Include Files” on page 1071.

**List of additional libraries to link with**
Enter the complete path of directories containing libraries with header files that are referenced by macros currently loaded into L-Edit.

**Update display**
When checked, L-Edit updates the display while UPI code is executing. When unchecked (default), L-Edit does not update the display during the execution of a macro or T-Cell generator.

**Show warning dialog boxes**
When checked (default), L-Edit displays warning dialog boxes in the user interface. When unchecked, L-Edit runs in *quiet mode*, in which warning dialog boxes are not displayed.

**Note:** Batch processing must be run in quiet mode.
UPI Include Files

The directory `<installdir>\Tanner Tools v xx.yy\upi\Include` contains a number of include files required by the L-Edit C++ interpreter.

The file `ldata.h` is a standard UPI include file that contains function prototypes of all UPI functions, including definitions of the “Interface Functions” (page 1075). The file `ldata.h` also contains definitions of all the supported hot key combinations.

The file `lcomp.h` is an optional UPI include file that contains function prototypes and inline documentation for a simple composition library that allows for chip assembly by relative object placement. The LComp interface is a streamlined interface to the ldata UPI procedures that minimizes the level of C language programming you need to use the UPI (see “LComp Functions Reference” on page 1811.) `lcomp.h` includes `ctype.h`, `malloc.h`, `math.h`, `stdarg.h`, `stdio.h`, `stdlib.h`, `string.h`, and `time.h`, which are L-Edit versions of standard C language header files.

Running a Macro

☑ Select **Tools > Macro** to open the **Macro** dialog.

☑ Click **Setup** to specify interpreter parameters (the locations of the header and log files).

☑ Click **Load** and select a macro file with a `.cpp` or `.dll` extension.

☑ Select the macro from the **Macros** list and click **Run**, or double click on the macro name in the dialog to execute it.

☑ Click **Edit** to open the active file (in this example, `mosfet.cpp`) in the L-Edit text editor.
This section provides a reference to the L-Edit User-Programmable Interface (UPI) and its C/C++ language functions and datatypes. Functions are grouped into three primary categories.

**Function Categories**

“**Interface Functions**” (page 1075) allow you to create interface elements such as dialogs, and to register UPI macros.

“**Database Functions**” (page 1153) allow you to create and manipulate a design database.

“**Data Types and Typedefs**” (page 1693) allow you to create and manipulate data structures.

**Data Relationships**

UPI functions operate on many different types of data. These data correspond to the various design components found within L-Edit (e.g., files, cells, drawing objects, selection lists, etc.). The following diagram shows the relationship between many of these components, with arrows indicating a “contains a” relationship, boxes showing data types, and boxes with drop shadows representing lists of items.
A primary user interface component is a “selection list”. In the L-Edit UPI, this implicitly corresponds to the selection list in the active cell.

Global Numerical Limits

Many UPI function variables must fall within a specified range of values to be valid in L-Edit. The following limits are pre-defined for all UPI functions:

- **WORLD_MAX**: The maximum positive value of an \(x\)- or \(y\)-coordinate in L-Edit.
- **MAX_LAYER_NAME**: The maximum number of characters in an L-Edit layer name.
- **MAX_CELL_NAME**: The maximum number of characters in an L-Edit cell name.
- **MAX_TDBFILE_NAME**: The maximum number of characters in an L-Edit file name.
Obsolete Functions

As the L-Edit database changes, data types and UPI functions must be updated or replaced. We strive to maintain backward compatibility but it is not always possible. Obsolete functions are marked as such in this documentation, and often the version number in which it was introduced is included in the name of a revised function.

Also, if a UPI program contains:

```c
#define EXCLUDE_LEDIT_LEGACY_UPI
```

```c
before #including any header files, then the obsolete UPI functions are not exposed to the user program.
```
Interface Functions

The “Dialog Functions” (page 1076) are used to create graphical interface elements such as dialog and message boxes.

The “Cursor and Display Functions” (page 1088) are used for tasks such as finding the current mouse position, displaying a message in the status bar, or getting and setting the visible cell in L-Edit’s layout window.

The “UPI Macro Functions” (page 1099) are used for loading, registering, and binding UPI macros.

The “UPI Functions” (page 1113) are used for finding the L-Edit serial number, choosing the selection tool, and inserting a menu item separator.

The “Windows Functions” (page 1128) are used to manipulate multiple windows and get window parameters in L-Edit.
Dialog Functions

“LDialog_MsgBox” (page 1077)  “LDialog_MultiLineMsgBox” (page 1078)
“LDialog_YesNoBox” (page 1080)  “LDialog_AlertBox” (page 1079)
“LDialog_InputBox” (page 1081)  “LDialog_MultiLineInputBox” (page 1082)
“LDialog_PickList” (page 1083)  “LDialog_File” (page 1084)

Dialog Button Returns

L-Edit UPI recognizes the following dialog button returns. Each is internally mapped to an integer value.

**LOK** Indicates that the **OK** button was selected.

**LCANCEL** Indicates that the **Cancel** button was selected.

**LYES** Indicates that the **Yes** button was selected.

**LNO** Indicates that the **No** button was selected.
LDialog_MsgBox

void LDialog_MsgBox(const char* szMessage)

Description

Produces a single-line message box.

Parameters

- **szMessage**: Specifies the message to be displayed.

Example

```c
LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    char szCellName[MAX_CELL_NAME];
    LCell_GetName(pCell, szCellName, sizeof(szCellName));

    LDialog_MsgBox(LFormat("Active Cellname = \"%s\".", szCellName));
}
```

See Also

- “Interface Functions” (page 1075), “LDialog_MultiLineMsgBox” (page 1078),
- “LDialog_AlertBox” (page 1079)
LDialog_MultiLineMsgBox

```c
void LDialog_MultiLineMsgBox( const char* szaMessages[], int nTotalEntries );
```

**Description**

Produces a multiple-line message dialog.

**Parameters**

- `szaMessages`: Array of strings. Each string will be displayed on a separate line in the dialog.
- `nTotalEntries`: Number of message lines.

**Example**

```c
LCell pCell = LCell_GetVisible();
LFile pTDBFile = LCell_GetFile(pCell);
if(Assigned(pCell) && Assigned(pTDBFile))
{
    char szCellName[MAX_CELL_NAME];
    LCell_GetName(pCell, szCellName, sizeof(szCellName));

    char szTDBFileName[MAX_TDBFILE_NAME];
    LFile_GetName(pTDBFile, szTDBFileName, sizeof(szTDBFileName));

    const char* szaMessages[2];
    szaMessages[0] = szCellName;
    szaMessages[1] = szTDBFileName;
    LDialog_MultiLineMsgBox(szaMessages, 2);

    szaMessages[0] = LFormat(“Active Cellname = \"%s\"." , szCellName);
    szaMessages[1] = LFormat(“Active TDB Filename = \"%s\"." ,
                        szTDBFileName);
    LDialog_MultiLineMsgBox(szaMessages, 2);
}
```

**See Also**

“Interface Functions” (page 1075), “LDialog_MsgBox” (page 1077), “LDialog_AlertBox” (page 1079)
LDialog_AlertBox

void LDialog_AlertBox( const char* szMessage );

Description

Produces a warning dialog.

Parameters

szMessage Warning displayed in the dialog.

Example

LFile pTDBFile = LFile_GetVisible();
if(Assigned(pTDBFile))
{
    char szTDBFileName[MAX_TDBFILE_NAME];
    LFile_GetName(pTDBFile, szTDBFileName, sizeof(szTDBFileName));

    LDialog_AlertBox(LFormat("Active TDB Filename = \"%s\".",
                        szTDBFileName));
}

See Also

“Interface Functions” (page 1075), “LDialog_MsgBox” (page 1077),
“LDialog_MultiLineMsgBox” (page 1078)
**LDialog_YesNoBox**

```c
int LDialog_YesNoBox( const char *msg );
```

**Description**

Produces a query dialog. One of two choices is clicked in response to the query.

**Return Values**

If **Yes** is clicked, the function returns **LYES**; if **No** is clicked, it returns **LNO**.

**Parameters**

*msg* Query to be displayed in the dialog.

**Example**

```c
if ( LDialog_YesNoBox("Do you want to Continue") ){
    /*Yes is clicked - the program continues*/
} else {
    /*No is clicked - the program exits*/
}
```

**See Also**

“Interface Functions” (page 1075), “Dialog Button Returns” (page 1076)
LDialog_InputBox

```c
int LDialog_InputBox( const char *title, const char *msg, char *ibuf );
```

Description

Produces an input dialog. The value entered by the user is returned as a string. If another datatype is needed, the string must be converted to the appropriate type.

Return Values

If OK is clicked, the function returns LOK; if Cancel is clicked, it returns LCANCEL.

Parameters

- `title` Title of the dialog box.
- `msg` Prompt displayed in the dialog.
- `ibuf` A buffer used to return the value entered at the prompt.

Example

```c
/*Allocate a buffer to store the return value*/
char value_buffer[50];

/*Initialize buffer to display a default value*/
strcpy(value_buffer, "pcell");

/*Display an input box with Cell Name Query as the title*/
if ( LDialog_InputBox("Cell Name Query", "Enter Name of the cell to be instanced", value_buffer) == 0)
    return;
```

See Also

"Interface Functions" (page 1075), “Dialog Button Returns” (page 1076)
LDialog_MultiLineInputBox

```c
int LDialog_MultiLineInputBox( const char *title, LDialogItem ibuf[], int total_entries );
```

**Description**

Produces a multiple-line input dialog. Several values are entered in response to prompts.

**Return Values**

If **OK** is clicked, the function returns **LOK**, if **Cancel** is clicked, then it returns **LCANCEL**.

**Parameters**

- `title` 
  Title of the dialog box.
- `ibuf` 
  Prompts displayed and the values entered.
- `total_entries` 
  Number of values expected.

**Example**

```c
/*Declare an array of dialog items to be displayed*/
LDialogItem  Dialog_Items [ 3 ] =
    { { "Inner Radius", "175" },
      { "Outer Radius", "200" },
      { "Teeth Count", "15" } };

long R_Inner, R_Outer, Teeth_Count;
float Teeth_Width;

/*Display a multi line dialog box to display the default values of gear
properties and get the modified properties*/
if ( LDialog_MultiLineInputBox ( "Gear Properties", Dialog_Items, 3 ) ){
    /* user selected OK, so get the property value from the Dialog_Items
buffer*/
    
    R_Inner = atol( Dialog_Items[0].value ); // get the inner radius
    R_Outer = atol( Dialog_Items[1].value ); // get the outer radius
    Teeth_Count = atol( Dialog_Items[2].value ); // get the teeth count
    /*Calculate Teeth Width*/
    Teeth_Width = 6.283185307 * R_Inner / ( 2 * Teeth_Count );
}
```

**See Also**

“LDialogItem” (page 1723), “Interface Functions” (page 1075), “Dialog Button Returns” (page 1076)
**LDialog_PickList**

```c
int LDialog_PickList( const char *title, const char *list[], int total_entries, int default_choice );
```

**Description**

Produces an input dialog. One of a list of possibilities is chosen either by highlighting the desired item and clicking **OK**, or by double-clicking the desired item.

**Return Values**

If **OK** is clicked (or the highlighted item is double-clicked), the function returns the index of the highlighted item; if **Cancel** is clicked, it returns –1.

**Parameters**

- `title`  
  Title of the dialog.
- `list`  
  Items listed.
- `total_entries`  
  Number of items listed.
- `default_choice`  
  Index of the item shown highlighted when the dialog first appears.

**Example**

```c
/*This example displays a pick list with three members to choose from*/

/*Declare a buffer to hold all elements of the pick list*/
char *Pick_List [ ] = {
    "Inverter",
    "Op-Amp",
    "Transistor"
};

/*Number of elements in the pick list*/
int Pick_Count = 3;

/*Index of the item picked by user*/
int Picked;

/*Display the pick list with Inverter as the default selection*/
Picked = LDialog_PickList ("Select Element", Pick_List, Pick_Count, 0);
```

**See Also**

“**Interface Functions**” (page 1075).
LDialog_File

```c
extern void LDialog_File( const char* szDefaultName, const char* szTitle, char* szFileNameBuffer, const char* szFiltersForBrowser, int iBrowseType, const char* szMessage, const char* szOkText, const char* szDefaultExt, const char* szTypeList, const LFile pFile );
```

**Description**

Call this function to display a filename edit box. The user may type a filename in the field provided, or click the **Browse** button to open a standard Windows file dialog. Clicking the **Browse** button opens a File Open or File Choose dialog box, depending on the value of `iBrowseType`.

The `szFilter` parameter is used to determine the type of filename a file must have to be displayed in the file list box. The first string in the string pair describes the filter; the second string indicates the file extension to use. Multiple extensions may be specified using a semicolon (;) as the delimiter. The string ends with two '|' characters, followed by a NULL character.

For example, L-Edit permits users to open files with extensions .TDB (layout), .OA or .VDB (LVS Results), among others. The filter for L-Edit could be written as:

```
"Layout Files (*.tdb)|*.tdb|LVS Files (*.vdb)|*.vdb|L-Edit Files (*.tdb;*.vdb)|*.tdb; *.vdb|All Files (*.*)|*.*|
```

**Return Values**

Name of the file (with a path). In the case that `TypeList` was provided, it returns as a string consisting of a file name and a 0 based index into the `TypeList`, separated with '|' character.

**Parameters**

- **DefaultName**
  - The initial filename that appears in the filename edit box. If NULL, no filename initially appears.

- **Title**
  - The title of the filename edit box.

- **FileNameBuffer**
  - Filename buffer to store the filename selected. If the user cancels this is set to the empty string.

- **FiltersForBrowser**
  - A series of string pairs that specify filters you can apply to the file. If you specify file filters, only selected files will appear in the Files list box. See the description section for more information on how to work with file filters. Strings are separated by the “|” character.

- **BrowseType**
  - Specifies the type of dialog opened when the user selects the Browse button in the filename edit box:
    - `iBrowseType` = 1 opens a **File Open** dialog.
    - `iBrowseType` ≠ 1 opens a **File Save** dialog.

- **Message**
  - Message to display in the filename edit box, such as instructions to the user.
OkText
Text displayed on the left-hand button in the filename edit box. The user clicks this button to accept the displayed filename.

DefaultExt
The default filename extension. If the user does not include an extension in the Filename edit box, the extension specified by szDefaultExt is automatically appended to the filename. If this parameter is NULL, no file extension is appended. Strings are separated by the “|” character. This field is only meaningful for the “Save” dialog.

TypeList
A series of strings that specify file types to be listed in the File Type: field of the filename edit box. This field is only meaningful for the “Save” dialog. If this parameter is set to NULL, the File Type: field is omitted from the filename edit box.

Each string in the series indicates a single file extension, ending with a ‘|’ character. The series ends with two ‘|’ characters, followed by a NULL character. For example,

cchar *szTypeList = "*.tdb|*.vdb|*.txt||"

includes the TDB, VDB, and TXT file extensions in the drop-down list of file types.

File
Pointer to an existing file. The location of pFile is the default directory for the File Open or File Choose dialog. When pFile is set, szFileNameBuffer contains the selected filename and path relative to the location of pFile. The pFile parameter is used to better support relative file names.

Example 1

LDIALOG_FILE( "tanner.tdb",
   "Export File",
   "Tanner Database Files (*.tdb)|*.tdb|GDSII Files (*.gds)|*.gds||", 0,
   "Current file will be exported to this file name:",
   "&Export",
   "tdb|gds||",
   "Tanner v15 Database File|GDSII File||",
   NULL
 );
// Return value of "C:\tmp\test.gds|1" will indicate that user selected GDSII file.

Example 2

/* Parameters for the filename edit box */
cchar* szFileName = "default_name";
const char* szTitle = "Open File";
char szFileNameBuffer[256];
const char* szMessage = "Please select a file to open:";
const char* szDefaultExt = "tdb";
const char* szTypeList = "*.tdb|*.vdb||";
const char* szOkText = "OK";

/* Type of Windows file dialog to open with the Browse button */
int iBrowseType = 2;

/* Parameters for the Windows file dialog */
const char* szFilterForBrowser = "Layout Files (*.tdb)|*.tdb|LVS Files
    (*.vdb)|*.vdb|L-Edit Files (*.tdb;*.vdb)|*.tdb; *.vdb|All Files
    (*.*)|*.*|");
LFile pFile = LFile_Find("C:/Tanner/LEdit90/Samples/T-Cells/alphabet.tdb");

/* Open the file edit box */
LDialog_File(szFileName, szTitle, szFileNameBuffer, szFilterForBrowser,
    iBrowseType, szMessage, szOkText, szDefaultExt, szTypeList, pFile);

The macro shown above opens the following filename edit dialog:

![Filename Edit Dialog](image)

If the user clicks Browse, L-Edit opens a standard file Open dialog (specified by \( \text{iBrowseType} = 1 \)) in the directory that contains \( \text{pFile} \).

![Open Dialog](image)

Example 3

```c
void LDialog_File(const char* szDefaultName, const char* szTitle, char* szFileNameBuffer,
    const char* szFiltersForBrowser, int iBrowseType, const char* szMessage,
    const char* szOkText, const char* szDefaultExt, const char* szTypeList,
    const LFile pFile);
```
// BrowseType meaning: 0 - "Save" dialog, 1 - "Open" dialog
// FiltersForBrowser, DefaultExt and TypeList are strings separated by '|
// character.
// Note: TypeList and DefaultExt are only meaningful for "Save" dialog.
// pFile parameter is used to better support relative file names.
//
// Returns: Name of the file (with a path). In case if TypeList was
// provided, it returns as string consisting of a
// file name and a 0 based index into the TypeList, separated with '|
// character.

const char szFileExt[] = "*.txt";
const char szFilter[] = "Text Files (*.txt)|*.txt|All Files (*.*)|*.*||";

char szTemp[512] = "\0";
LDIALOG_FILE("MyFile.txt", // Default filename. This is the string that
is put in the browse dialog including path.
"Please select the Batch Rendering Script file.", // Dialog title
szTemp, // Variable to store the filename
szFilter, // File type file.
1, // Browse Type - 1 Open, 0 - Save As.
"Batch Rendering Script filename:" // Filename message in the Dialog.
NULL, // String for the OK button - NULL means OK.
szFileExt, // Default extension type.
NULL, // Type list - a list of extension types to indicate which type was
selected.
NULL); // LFile pointer - if you send a LFile to LDialog_File, then all
relative paths will be relative to the TDB referenced by LFile.

if(strlen(szTemp) > 0)
{
    // The user did not cancel the dialog.
    // Do my processing here.
}

// Another example
// Example:
// LDIALOG_FILE("tanner.tdb",
// "Export File",
// "Tanner Database Files (*.tdb)|*.tdb|GDSII Files (*.gds)|*.gds||",
// 0,
// "Current file will be exported to this file name:",
// "&Export",
// "tdb|gds||",
// "Tanner v8 Database File|GDSII File||",
// NULL
//);
// Return value of "C:mpest.gds|1" will indicate that user selected GDSII
file.

See Also

"LFile_GetResolvedFileName" (page 1244)
Cursor and Display Functions

- “LCursorGetPosition” (page 1089) (Obsolete)
- “LCursorGetPositionEx99” (page 1090)
- “LCursorGetSnappedPosition” (page 1091)
- “LDisplayRefresh” (page 1092)
- “LStatusBarSetMsg” (page 1093)
- “LCellHomeView” (page 1094)
- “LCellGetVisible” (page 1095)
- “LCellGetLastVisible” (page 1096)
- “LCellMakeVisible” (page 1097)
- “LCellMakeVisibleNoRefresh” (page 1098)
**LCursor_GetPosition**

```c
LPoint LCursor_GetPosition( void );
```

**Description**

Gets the current cursor (mouse) position.

**Return Value**

Returns the current cursor (mouse) position.

**Example**

```c
LPoint pt = LCursor_GetPosition();
LDialog_MsgBox( LFormat( "( %ld, %ld )", pt.x, pt.y ) );
```

**Version**

As of L-Edit V9, this function has been deprecated, in favor of “LCursor_GetPositionEx99” (page 1090)

**See Also**

**LCursor_GetPositionEx99**

```c
LPoint LCursor_GetPositionEx99(  
    int iSnapped,  
    int iPauseForInput,  
    const char* szMessage )
```

**Description**

Gets the current cursor (mouse) position. Optionally the cursor position can be snapped to the current snap grid settings in the **Setup > Design—Grid** tab. LCursor_GetPositionEx99 gets the current cursor position and immediately returns. One can optionally pause for user input, allowing the user to press the left mouse button to indicate the cursor position.

**Return Value**

Returns the current cursor (mouse) position.

**Parameters**

- **iSnapped**
  
  Snap the cursor position to the current snap settings (1 = True, 0 = False).

- **iPauseForInput**
  
  Pause so the user can press the mouse left button to indicate the cursor position (1 = True, 0 = False).

- **szMessage**
  
  Displays the message when pausing for user input. If `szMessage` is NULL, then it displays “Please pick a point.”

**Example**

```c
// get a point from the user, and print out its coordinates
LPoint pt = LCursor_GetPositionEx99( true, true, "click desired point" );
LDialog_MsgBox( LFormat( "%ld, %ld ", pt.x, pt.y ) );
```

**See Also**

“**LCursor_GetPosition**” (page 1089), “**Interface Functions**” (page 1075).
**LCursor_GetSnappedPosition**

```c
LPoint LCursor_GetSnappedPosition( void )
```

**Description**

Gets the current snapped cursor (mouse) position.

**Return Values**

Returns the current snapped cursor (mouse) position as a `LPoint`.

**Version**

As of L-Edit V9, this function has been deprecated, in favor of **“LCursorGetPositionEx99”** (page 1090).

**See Also**

“LTransform” (page 1801), “Interface Functions” (page 1075)
LDisplay_REFRESH

void LDisplay_REFRESH( void );

Description

Updates the display to show layout modifications produced by UPI calls.

See Also

“Interface Functions” (page 1075)
LStatusBar_SetMsg

void LStatusBar_SetMsg( const char *msg );

Description

Displays a message in the status bar. To clear the status bar, set msg to "" (an empty string).

Parameters

msg Message to be displayed.

Example

int nBoxes = 5;
LStatusBar_SetMsg( LFormat( "Boxes processed = %d", nBoxes ) );

See Also

“Interface Functions” (page 1075)
**LCell_HomeView**

```c
LStatus LCell_HomeView( LCell cell );
```

**Description**

Displays the home view of a given cell.

**Return Values**

LStatusOK if successful. If an error occurs, `LStatus` contains the error value.

**Parameters**

- `cell` Specified cell.

**Example**

```c
// zoom to home view and refresh display
LCell pCell = LCell_GetVisible();
if ( pCell )
{
    LCell_HomeView( pCell );
    LDisplay_Refresh();
}
```

**See Also**

LCell_GetVisible

LCell LCell_GetVisible( void );

Description

Gets the visible (active) cell in the layout window.

Return Values

Returns a pointer to the active cell; otherwise NULL.

Example

LCell pCell = LCell_GetVisible();
if ( pCell )
{
    // print the name of the active cell
    char buf[ MAX_CELL_NAME ];
    LDialog_MsgBox(
        LFormat( "Active cell is %s",
                LCell_GetName( pCell, buf, sizeof( buf ) )
        )
    );
}

See Also

“LStatus” (page 1792), “Interface Functions” (page 1075)
LCell_GetLastVisible

LCell LCell_GetLastVisible( LFile file );

Description

Gets the cell last open in the specified file.

Return Values

Returns a pointer to the last open cell in the specified file, or NULL on error.

Parameters

file Specified file.

See Also

“LObject” (page 1769), “Interface Functions” (page 1075)
LCell_MakeVisible

LStatus LCell_MakeVisible( LCell cell );

Description

Makes the specified cell the current one and updates the display.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell

Specified cell.

See Also

“LStatus" (page 1792), “LObject" (page 1769), “Interface Functions” (page 1075)
LCell_MakeVisibleNoRefresh

LStatus LCell_MakeVisibleNoRefresh( LCell cell );

Description

Makes the specified cell the current one without updating the display.

Return Values

If an error occurs, it returns LBadCell; otherwise returns LStatusOK.

Parameters

 cell Specified cell.

See Also

“LStatus” (page 1792), “LObject” (page 1769), “Interface Functions” (page 1075)
UPI Macro Functions

- “LMacro_Register” (page 1100)
- “LMacro_BindToHotKey” (page 1101)
- “LMacro_BindToMenu” (page 1102)
- “LMacro_BindToMenuAndHotKey_v9_30” (page 1103)
- “LMacro_IsLoaded” (page 1108)
- “LMacro_Load” (page 1109)
- “LMacro_LoadEx1200” (page 1110)
- “LMacro_UnLoad” (page 1111)
- “LMacro_GetNewTCell” (page 1112)
## LMacro_Register

```c
void LMacro_Register( char *macro_desc, char *function );
```

### Description

Registers a user-defined macro in L-Edit.

### Parameters

- **macro_desc**
  - Macro name that should be displayed in the *Macros list*.
- **function**
  - Name of the macro function.

### Example

```c
void macro_function( void )
{
    // macro code goes here
}

void UPI_Entry_Point( void )
{
    LMacro_Register( "My macro’s description", "macro_function" );
}
```

### See Also

- “*Interface Functions*” (page 1075), “*Running a Macro*” (page 1071)
**LMacro_BindToHotKey**

```c
void LMacro_BindToHotKey( int keycode, char *macro_desc, char *function);
```

**Description**

Establishes a relationship between a user-defined macro and a keyboard shortcut ("hot key") so that user can invoke the macro by pressing the hot key. Supported key combinations (keycodes) are defined in `<install_dir>/include/lupi_usr.h`.

**Parameters**

- `keycode`  
  Keyboard shortcut (for example, `KEY_F2` for the `F2` key).
- `macro_desc`  
  String displayed in the **Macros** list of the **Run Macro** dialog.
- `function`  
  Macro function name.

**Note:**  
Note that this function is superseded in L-Edit V10 and later.

**See Also**

“Interface Functions” (page 1075), “Binding Macros to Hot Keys” (page 1049)
**LMacro_BindToMenu**

```c
void LMacro_BindToMenu( char *menu, char *macro_desc, char *function );
```

**Description**

Establishes a relationship between a user-defined macro and a menu command. When the menu item is selected, the macro is executed.

**Parameters**

- **menu**: Main L-Edit menu in which to add the entry (for example, **Tools**).
- **macro_desc**: Menu entry to add. If this string begins with a space, the menu entry is preceded by a separator line. A submenu may be specified by putting a `\` in the string (for example, "My Program\My Macro"). The macro_desc string is also displayed in the **Macros** list of the Run Macro dialog.
- **function**: Macro function name.

**Example**

```c
void macro1( void )
{
    // macro code goes here
}

void UPI_Entry_Point( void )
{
    LMacro_BindToMenu( "Tools", "My macro", "macro1" );
    LMacro_BindToMenu( "Tools", "Subdir\My macro", "macro1" );
    LMacro_BindToMenu( "Tools", "Subdir\My macro2", "macro2" );
}
```

**Note:**

Note that this function is superseded in L-Edit V10 and later.

**See Also**

“**Interface Functions**” (page 1075), “**Binding Macros to Menu Items**” (page 1049)
**LMacro_BINDToMenuAndHotKey_v9_30**

```
unsigned int LMacro_BindToMenuAndHotKey_v9_30(const char* szMenu, const char* szHotKey, const char* szMacroDescription, const char* szFunctionName, const char* szHotKeyCategory)
```

**Description**

Establishes a relationship between a user-defined macro and a menu command and/or a hotkey. When the menu item or the hotkey is selected, the macro is executed.

**Return Values**

If an error occurs, it returns 0; otherwise returns the resource ID number of the binding.

**Parameters**

- **szMenu**
  Main L-Edit menu in which to add the entry (for example, **File** or **Edit** or **Tools**). If this is **NULL**, then the macro will not be assigned to a menu.

- **szHotKey**
  A string of the hotkey to bind the macro to. The string is the actual key just like in **Setup>Application** dialog - **Keyboard** tab. The following modifiers and special keys are also available:
  - **Shift** - Indicates the Shift key is pressed in conjunction with another key.
  - **Ctrl** - Indicates the Control key is pressed in conjunction with another key.
  - **Alt** - Indicates the Alt key is pressed in conjunction with another key.
  - **F1** - Indicates F1 function key.
  - **Num 0** - Indicates the zero key on the numeric key pad.
  - **Backspace** - The backspace key.
  - **Tab** - The Tab key.
  - **Space** - The spacebar.
  - **Page Up** - The page up key.
  - **Page Down** - The page down key.
  - **End** - The end key.
  - **Home** - The home key.
  - **Left** - The left arrow.
  - **Up** - The up arrow.
  - **Right** - The right arrow.
  - **Down** - The down arrow.
  - **Ins** - The insert key.
  - **Del** - The delete key.
  - **Num *** - The * key on the numeric key pad.
  - **Num +** - The + key on the numeric key pad.
  - **Num -** - The - key on the numeric key pad.
  - **Num Del** - The Delete/decimal key on the numeric key pad.
  - **Num Lock** - The Num Lock key.
  - **Scroll Lock** - The * key on the numeric key pad.
  - **Break** - The Break key.
  - **Esc** - The escape key.
  - **Comma** - The , key.
**Period** - The . key.
To assign shifted keys such as @ or <, the use Shift and the lowercase key (for Example: @ is Shift+2 and < is Shift+Comma).
If this is NULL, then the macro will not be assigned a hotkey.

\[
\text{szMacroDescription}
\]
Menu entry to add. If this string begins with a space, the menu entry is preceded by a separator line. A submenu may be specified by putting a \ in the string (for example, "My Program\My Macro").

The \text{szMacroDescription} string is also displayed in the Macros list of the Tools>Macro dialog. If you want to place the macro at a specific location in a menu then add a newline \ and the full path to the menu item that you want your macro to be before (for example: My Macro\Edit\Copy will put the My Macro menu item between Cut and Copy on the Edit menu).
If the full path ends in \ then the menu item will be placed at the end of the menu (for example: `My Macro\View\Zoom\` will put the **My Macro** menu item at the end of the **Zoom** submenu after **Zoom to Selections**).

If the menu item has a separator before it and if you precede the menu item with a space, then it will put your macro before the separator, otherwise it will put it after the separator (for example: `My Macro\ View\ Home` will put the **My Macro** menu item on the **View** menu before the separator that is above the **Home** menu item).

You can also add a new menu at the top level by indicate a top level menu to place your macro before (for example: `My Program\My Macro\Tools` puts the **My Program** menu item before the **Tools** menu).

```plaintext
szFunctionName
```

Macro function name.
**szHotKeyCategory**

This is a string of the category that the hotkey should be in the Setup>Application dialog - Keyboard tab for remapping hotkeys (For Example, Macro or Tools or Cell). If this is NULL, then the category will be Macro. The category does not have to be an existing category (for example: My Macros category would create a new category).

### Example

```c
void Macro1(void)
{
    // macro code goes here
}

void UPI_Entry_Point(void)
{
    LMacro_BindToMenuAndHotKey_v9_30("Edit",
        "F1",
        "My Program\My Macro",
        "Macro1",
        "My Macros");

    LMacro_BindToMenuAndHotKey_v9_30("Edit",
        "Shift+2",
        "My Macro\nEdit\nCopy",
        "Macro2",
        NULL);

    LMacro_BindToMenuAndHotKey_v9_30("View",
        NULL,
        "My Macro\nView\Zoom\",
        "Macro3",
        NULL);

    LMacro_BindToMenuAndHotKey_v9_30("View",
        "Shift+Comma",
        "My Macro\nView\Home",
        "Macro4",
        NULL);
```
LMacro_BindToMenuAndHotKey_v9_30(NULL, "End", "My Program\My Macro\nTools", "Macro5", NULL);

}  

Version

Available in L-Edit v10.0 and later versions.

See Also

“Interface Functions” (page 1075), “Binding Macros to Menu Items” (page 1049)
LMacro_IsLoaded

    int LMacro_IsLoaded( char *dll_path );

Description

Indicates whether a DLL macro is loaded.

Return Values

1 if macro is loaded, 0 if not.

Parameters

    dll path              Full path and file name of the macro.

Example

    if ( LMacro_IsLoaded("c:\keyhole.dll") )
        LMacro_UnLoad("c:\keyhole.dll");

Version

Available in L-Edit 8.4 and later versions.

See Also

“LMacro_Load” (page 1109), “LMacro_UnLoad” (page 1111).
**LMacro_Load**

```c
int LMacro_Load( char *dll_path );
```

**Description**

Loads a DLL macro.

**Return Values**

1 if the macro loads successfully, 0 if not.

**Parameters**

- `dll_path` Full path and file name of the macro.

**Example**

```c
LMacro_Load("c:\\keyhole.dll");
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

LMacro_LoadEx1200

```c
int LMacro_LoadEx1200(const char* szDLL_Path, LBoolean bLoadAtStartUp);
```

**Description**

Loads a DLL macro and indicates whether to load that macro at startup.

**Return Values**

1 if the macro loads successfully, 0 if not.

**Parameters**

- `szDLL_Path` Full path and file name of the macro DLL to load.
- `bLoadAtStartUp` LTRUE to load the macro when L-Edit starts up.

**Example**

```c
LMacro_LoadEx1200("c:\\keyhole.dll", LTRUE);
```

**Version**

Available in L-Edit 12.00 and later versions.

**See Also**

**LMacro_UnLoad**

```c
void LMacro_UnLoad( char *dll_path );
```

**Description**

Unloads a DLL macro.

**Parameters**

*dll path*  
Full path and file name of the macro.

**Example**

```c
LMacro_UnLoad("c:\\keyhole.dll");
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LMacro_IsLoaded” (page 1108), “LMacro_Load” (page 1109)
LMacro_GetNewTCell


```
LMacro_GetNewTCell( void );
```

**Description**

Gets a handle to the auto-generated cell created by running T-Cell generator code. L-Edit automatically generates a call to `LMacro_GetNewTCell` when it creates a T-Cell generator code template. Normally, you will not need to add calls to this function.

**Return Values**

Returns a pointer to the auto-generated cell if successful. Otherwise, returns NULL.

**Version**

Available in L-Edit v9.0 and later versions.

**See Also**

UPI Functions

- “LUpi_GetSerialNumber” (page 1114)
- “LUpi_LogMessage” (page 1115)
- “LUpi_SetQuietMode” (page 1116)
- “LUpi_InQuietMode” (page 1117)
- “LUpi_SetSelectionTool” (page 1118)
- “LUpi_SetDrawingTool” (page 1119)
- “LUpi_InsertMenuItemSeparator” (page 1120)
- “LUpi_SetReturnCode” (page 1121)
- “LUpi_GetReturnCode” (page 1122)
- “LUpi_SetUpdateDisplayMode” (page 1124)
- “LUpi_GetUpdateDisplayMode” (page 1125)
- “LFormat” (page 1126)
- “LFormatV” (page 1127)
**LUpi_GetSerialNumber**

```c
long LUpi_GetSerialNumber( void );
```

**Description**

Gets the serial number of L-Edit.

**Return Values**

Returns the serial number or -1 on error.

**Example**

```c
#include <set>
#include <ldata.h>
using namespace std;

extern "C" {
    void HelloWorld(void);
    int UPI_Entry_Point(void);
}

void HelloWorld(void)
{
    LCell    pCell    =    LCell_GetVisible();
    LFile    pFile    =    LCell_GetFile(pCell);
    LDialog_MsgBox("Hello World");
}

int UPI_Entry_Point(void)
{
    //Hardware key IDs in HEX. Only keys with these numbers are allowed to run the HelloWorld macro.
    long key_arr[] = {0x4259CE, 0x1026C49};

    set<long> KeySet;
    set<long>::iterator it;

    for (int i=0; i < sizeof(key_arr)/sizeof(key_arr[0]); i++)
        KeySet.insert(key_arr[i]);

    //Get the HEX ID of the key attached to the computer.
    long lSerial = LUpi_GetSerialNumber();

    it = KeySet.find(lSerial); //Search for this HEX ID in the set.

    //If found, then execute the HelloWorld Macro.
    if (it != KeySet.end())
        LMacro_BindToMenuAndHotKey_v9_30("Tools", NULL /*hotkey*/,
                                            "Hello World", NULL /*hotkey category*/);
    else
        LDialog_MsgBox("Key not found");

    return 1;
}
```
LUPI_LogMessage

```c
void LUPI_LogMessage( const char *msg );
```

**Description**

Displays a message in the log window.

**Parameters**

- `msg` Message to be displayed.

**Example**

```c
LUPI_LogMessage("hello world");
```

**See Also** "Interface Functions" (page 1075)
**LUpi_SetQuietMode**

```c
LStatus LUpi_SetQuietMode( int val );
```

**Description**

Sets the quiet mode. When the quiet mode is on, the alert boxes are suppressed. The use of quiet mode is required for batch processing.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `val` Quiet mode control (1=ON, 0=OFF).

**See Also**

“LUpi_InQuietMode” (page 1117), “Interface Functions” (page 1075)
LUpi_InQuietMode

```c
int LUpi_InQuietMode( void );
```

**Description**

Gets the quiet mode. When the quiet mode is on, the alert boxes are suppressed. The use of quiet mode is required for batch processing.

**Return Values**

1 if quiet mode is on, 0 if quiet mode is off

**Example**

```c
int bQM = LUpi_InQuietMode();
// now turn off all prompts
LUpi_SetQuietMode( true );
// do the macro...
... user code goes here...
// reset original mode
LUpi_SetQuietMode( bQM );
```

**See Also**

“LUpi_SetQuietMode” (page 1116), “Interface Functions” (page 1075)
**LUpi_SetSelectionTool**

```c
void LUpi_SetSelectionTool( void );
```

**Description**

Selects the selection tool in L-Edit.

**See Also**

"Interface Functions" (page 1075), "Running a Macro" (page 1071)
**LUpi_SetDrawingTool**

```
UPIDrawingToolType LUpi_SetDrawingTool( UPIDrawingToolType eTool );
```

**Description**

Selects the specified drawing tool.

**Return Values**

Returns the previously selected drawing tool.

**Parameters**

- **eTool**
  The drawing tool to select.

**See Also**

“*Interface Functions*” (page 1075), “*UPIDrawingToolType*” (page 1809)
LUpi_InsertMenuItemSeparator

```c
void LUpi_InsertMenuItemSeparator( char *menu );
```

Description

Appends a separator in the specified L-Edit menu. This function can be used for separating menu items.

Parameters

- **menu**
  
  Name of menu where separator is to be inserted.

See Also

“Interface Functions” (page 1075), “Running a Macro” (page 1071)
LUpi_SetReturnCode

```c
void LUpi_SetReturnCode( int nCode );
```

**Description**

Sets the UPI return code, which is global to L-Edit. It is reset to zero just before an outermost macro is executed. An outermost macro is a macro that is not nested within another.

**Note:**

A nested macro, such as an LC_Generate call within a T-Cell generator, will execute its code *without* resetting the UPI return code.

The T-Cell mechanism in L-Edit checks the UPI return code after executing generator code for a T-Cell. If the UPI return code is nonzero, L-Edit destroys the auto-generated cell and no instance is created.

**Parameters**

- **nCode**
  The new value of the UPI return code.

**Example**

```c
/* This is an excerpt from a T-Cell generator code function. */
char* layername = "Polka Dot";
LFile myFile = LFile_GetVisible();
LLayer mylayer = LLayer_Find( myFile, layername );
if (mylayer == (LLayer)NULL)
{
    LUpi_SetReturnCode(-1);
    LDialog_MsgBox( LFormat("Could not find layer %s, cell not created.", layername) );
    return;
}
```

**Version**

Available in L-Edit 9.0 and later versions.

**See Also**

“LUpi_GetReturnCode” (page 1122)
LUpi_GetReturnCode

```c
int LUpi_GetReturnCode( void );
```

Description

Gets the UPI return code, which is global to L-Edit. It is reset to zero just before an outermost macro is executed. An outermost macro is a macro that is not nested within another.

**Note:** A nested macro, such as an LC_Generate call within a T-Cell generator, will execute its code *without* resetting the UPI return code.

The T-Cell mechanism in L-Edit checks the UPI return code after executing generator code for a T-Cell. If the UPI return code is nonzero, L-Edit destroys the auto-generated cell and no instance is created.

Return Values

The value of the UPI return code.

Example

```c
/* Check for a valid UPI return code before instancing a T-Cell. */
#include "lcomp.h"

void MetaGen_main(void)
{
    /* Parameter variables */
    LCell    cellCurrent;
    LLayer   MetaLayer;

    /* Other variables */
    char     szLayerName[128];
    char*    params[10];    /* array of pointers to character strings */

    /* Initialize parameter variables */
    cellCurrent = (LCell)LMacro_GetNewTCell();
    MetaLayer= (LLayer)LCell_GetParameter(cellCurrent, "MetaLayer");

    /* Initialize L-Comp */
    LC_InitializeState();
    LC_CurrentCell = cellCurrent;

    LLayer_GetName( MetaLayer, szLayerName, sizeof(szLayerName) );
    /* parameter 1, name and value */
    params[0] = "BoxLayer";params[1] = szLayerName;
    /* end parameter list with NULL */
    params[2] = NULL;
    LC_Generate( "BoxGen", "My Auto Box", params );
    if (LUpi_GetReturnCode() != 0)
        return;
    /* more layout here */
}```
Version

Available in L-Edit 9.0 and later versions.

See Also

“LUpi_SetReturnCode” (page 1121)
LUpi_SetUpdateDisplayMode

LStatus LUpi_SetUpdateDisplayMode( int val );

Description

Sets the update display mode for UPI. The update display mode determines whether the display is updated during the execution of a macro or T-Cell generator.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

val Value of the update display mode. There are two possible states for the update display mode:

- When the update display mode is zero, L-Edit’s display is not updated during the execution of a macro or T-Cell generator.
- When the update display mode is nonzero, the display is updated during macro execution.

Version

Available in L-Edit 9.0 and later versions.

See Also

“LUpi_GetUpdateDisplayMode” (page 1125), “LStatus” (page 1792)
LUpi_GetUpdateDisplayMode

int LUpi_GetUpdateDisplayMode( void );

Definition

Gets the current UPI update display mode. The update display mode determines whether the display is updated during the execution of a macro or T-Cell generator.

Return Values

Returns the current display mode. When the update display mode is zero, L-Edit’s display is not updated during the execution of a macro or T-Cell generator. When it is nonzero, the display is updated during macro execution.

Version

Available in L-Edit 9.0 and later versions.

See Also

“LUpi_SetUpdateDisplayMode” (page 1124)
**LFormat**

```c
const char* LFormat( const char* lpszFormat, ... );
```

**Description**

*LFormat* is similar to *sprintf*, but allocates and returns the storage for results. L-Edit deallocates this storage automatically when the macro or T-Cell generator code finishes execution.

**Return Values**

Returns a pointer to the resulting text string if successful; otherwise, returns NULL.

**Parameters**

The first parameter, *lpszFormat*, is a format specification string. This is followed by the list of variables whose values are to be inserted in the *lpszFormat* string.

**Example**

```c
/* Generate a message using LFormat */
int Count = 5;
const char *Name = "Boxes";
const char *Msg = LFormat("There are %d %s.", Count, Name);
LDialog_MsgBox(Msg);
```

**Version**

Available in L-Edit 9.0 and later versions.
LFormatV

const char* LFormatV(const char* lpszFormat, char** argList);

Description

LFormatV is similar to vsprintf, but allocates and returns the storage for results. L-Edit deallocates this storage automatically when the macro or T-Cell generator code finishes execution.

Return Values

Returns a pointer to the resulting text string if successful; otherwise, returns NULL.

Parameters

The first parameter, lpszFormat, is the format specification string.

The second parameter, argList, is an object representing the variable arguments list.

Example

void PrintfError(char* szFormat, ...)  
{  
    va_list args;  
    va_start(args, szFormat);  
    LDIALOG_MsgBox(LFormatV(szFormat, args));  
    va_end (args);  
}

Version

Available in L-Edit 9.0 and later versions.
Windows Functions

- “LWindow_GetVisible” (page 1129)
- “LWindow_GetList” (page 1130)
- “LWindow_GetNext” (page 1131)
- “LWindow_IsLast” (page 1132)
- “LWindow_MakeVisible” (page 1133)
- “LWindow_Close” (page 1134)
- “LWindow_CloseAll” (page 1135)
- “LWindow_EditInPlacePushIn” (page 1136)
- “LWindow_EditInPlacePopOut” (page 1137)
- “LWindow_EditInPlacePopToTop” (page 1138)
- “LWindow_GetType” (page 1139)
- “LWindow_GetFile” (page 1140)
- “LWindow_GetCell” (page 1141)
- “LWindow_GetEditTransform” (page 1142)
- “LWindow_GetTopCell” (page 1143)
- “LWindow_GetParameters” (page 1144)
- “LWindow_GetWindowHandle” (page 1145)
- “LWindow_NewTextWindow” (page 1146)
- “LWindow_LoadTextFile” (page 1147)
- “LWindow_SaveToFile” (page 1148)
- “LWindow_GetText” (page 1149)
- “LWindow_SetText” (page 1150)
- “LWindow_GetName” (page 1151)
- “LWindow_SetName” (page 1152)
- “LWindow_SetText” (page 1150)
- “LWindow_GetName” (page 1151)
- “LWindow_SetName” (page 1152)
LWindow_GetVisible

LWindow LWindow_GetVisible( void )

Description

Retrieves the active window.

Return Values

Returns an LWindow pointer to the active window; otherwise NULL.

Example

LWindow pWindow = LWindow_GetVisible();
if( NotAssigned( pWindow ) )
{
    LDialog_AlertBox( "No windows are active." );
    return;
}

See Also

“Interface Functions” (page 1075), “Windows Functions” (page 1128), “LWindow” (page 1804),
**LWindow_GetList**

```
LWindow LWindow_GetList( void )
```

**Description**

Retrieves the first open window.

**Return Values**

Returns an `LWindow` pointer to the first open window. If no windows are open, returns `NULL`.

**Example**

```
// Count the number of layout windows that are open.
long nNumOfLayoutWindows = 0;
LWindow pWindow = NULL;
for( pWindow = LWindow_GetList(); Assigned(pWindow);
    pWindow = LWindow_GetNext(pWindow) )
{
    if( (LWindow_GetType(pWindow) == LAYOUT) ||
        (LWindow_GetType(pWindow) == CROSS_SECTION) )
    {
        nNumOfLayoutWindows++;
    }
}
```

**Version**

Available in L-Edit 9.0 and later versions.

**See Also**

LWindow_GetNext

LWindow LWindow_GetNext( LWindow pWindow )

Description

Retrieves the next open window.

Return Values

Returns a pointer to the next open window. If pWindow is the last open window, returns NULL.

Parameters

pWindow Pointer to an L-Edit window.

Example

// Count the number of layout windows that are open.
long nNumOfLayoutWindows = 0;
LWindow pWindow = NULL;
for( pWindow = LWindow_GetList(); Assigned(pWindow);
    pWindow = LWindow_GetNext(pWindow) )
{
    if( (LWindow_GetType(pWindow) == LAYOUT) ||
        (LWindow_GetType(pWindow) == CROSS_SECTION) )
    {
        nNumOfLayoutWindows++;
    }
}

Version

Available in L-Edit 9.0 and later versions.

See Also

LWindow_IsLast

```c
int LWindow_IsLast( LWindow pWindow )
```

Description

Indicated if `pWindow` is the only currently open window in L-Edit.

Return Values

Returns 1 if `pWindow` is the only currently open window in L-Edit, 0 otherwise.

Parameters

- `pWindow` Pointer to an L-Edit window.

Example

```c
// Close all windows except one.
LWindow pWindow = LWindow_GetVisible();
while( Assigned(pWindow) )
{
    if( LWindow_IsLast(pWindow) == 0 )
    {
        LWindow_Close( pWindow );
    }
    else
    {
        break;
    }

    pWindow = LWindow_GetVisible();
}
```

See Also

“Interface Functions” (page 1075), “Windows Functions” (page 1128), “LWindow” (page 1804)
LWindow_MakeVisible

LStatus LWindow_MakeVisible ( LWindow pWindow )

Description

Sets the active window.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

LBadParameter – pWindow is NULL.

Parameters

pWindow Pointer to an L-Edit window.

Example

// Make the first layout window active.
LWindow pWindow = NULL;
for( pWindow = LWindow_GetList(); Assigned(pWindow);
    pWindow = LWindow_GetNext(pWindow) )
{
    if( (LWindow_GetType(pWindow) == LAYOUT) ||
        (LWindow_GetType(pWindow) == CROSS_SECTION) )
    {
        LWindow_MakeVisible(pWindow);
        break;
    }
}

See Also

LWindow_Close


    LStatus LWindow_Close( LWindow pWindow );

Description

Closes the specified window.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

LBadParameter - pWindow is NULL.

Parameters

pWindow Window to be closed.

Example

    // Close the active window. 
    LWindow pWindow = LWindow_GetVisible();
    if( Assigned( pWindow ) )
    {
        LWindow_Close( pWindow );
    }

See Also

LWindow_CloseAll

\[ \text{LStatus } \text{LWindow\_CloseAll( void )} \]

**Description**

Closes all open windows.

**Return Values**

\[ \text{LStatusOK} \]

**Example**

```c
// Close all windows.
LWindow_CloseAll();
```

**See Also**

LWindow_EditInPlacePushIn

LStatus LWindow_EditInPlacePushIn( LWindow pWindow, LInstance pInstance );

Description

Set the instance that is being edited in place.
**LWindow>EditInPlacePopOut**

```c
LStatus LWindow_EditInPlacePopOut( LWindow pWindow );
```

**Description**

Pop out the current editing level.
LWindow_EditInPlacePopToTop

LStatus LWindow_EditInPlacePopToTop( LWindow pWindow );

Description

Pop all the way to the top of the edit-in-place stack.
LWindow_GetType

LWindowType LWindow_GetType( LWindow pWindow )

Description

Retrieves the window type.

Return Values

Returns the type of the specified window.

Parameters

$pWindow$ Pointer to an L-Edit window.

Example

// Count the number of layout windows that are open.
long nNumOfLayoutWindows = 0;
LWindow pWindow = NULL;
for( pWindow = LWindow_GetList(); Assigned(pWindow); pWindow =
    LWindow_GetNext(pWindow) )
{
    if( (LWindow_GetType(pWindow) == LAYOUT) ||
        (LWindow_GetType(pWindow) == CROSS_SECTION) )
    {
        nNumOfLayoutWindows++;
    }
}

See Also

“Interface Functions” (page 1075), “Windows Functions” (page 1128), “LWindowType” (page 1805),
“LWindow” (page 1804)
LWindow_GetFile

LFile LWindow_GetFile( LWindow pWindow );

Description

Retrieves the TDB file associated with a window.

Return Values

Returns a pointer to the TDB file of the specified layout window. If the window is not a layout, cross-section, or Design Navigator window, then NULL is returned.

Parameters

pWindow Pointer to an L-Edit window.

Example

// Get TDB file of the active window.
LWindow pWindow = LWindow_GetVisible();
if( Assigned( pWindow ) )
{
    LFile pTDBFile = LWindow_GetFile( pWindow );
    if( Assigned( pTDBFile ) )
    {
        // More Processing on the edit cell of the active window.
        // ...
    }
}

See Also

**LWindow_GetCell**

```c
LCell LWindow_GetCell( LWindow pWindow )
```

**Description**

Retrieves the edit cell of a window. This is used to get the current cell that is being edited if the user is editing in place in this window.

**Return Values**

Returns a pointer to the edit cell of the specified layout window. If the window is not a layout window, then **NULL** is returned.

**Example**

```c
// Get the edit cell of the active window.
LWindow pWindow = LWindow_GetVisible();
if( Assigned( pWindow ) )
{
    LCell pCell = LWindow_GetCell( pWindow );
    if( Assigned( pCell ) )
    {
        // More Processing on the edit cell of the active window.
        // ...
    }
}
```

**Parameters**

- `pWindow` Pointer to an L-Edit window.

**See Also**

LWindow_GetEditTransform

LStatus LWindow_GetEditTransform( LWindow pWindow, LTransform_Ex99 *pEditTransform );

Description

Retrieves the edit transform for the specified window, if that window is a layout window. The edit transform is the transformation from the cell, which is being edited to the top-level cell. If the user is not editing in place in this window, then the edit transform will be the unit transform.

Return Values

If no errors occur, it returns LStatusOK. Returns LBadParameters if pWindow or pEditTransform is NULL or pWindow is not a layout window.

Example

Parameters

pWindow Pointer to an L-Edit layout window.
pEditTransform Pointer to the transform which will be set to the edit transform of the specified window.

Version

Available in L-Edit 10.1 and later versions.

See Also

**LWindow_GetTopCell**

```c
LCell LWindow_GetTopCell( LWindow pWindow )
```

**Description**

Retrieves the top-level cell of a window. This is used to get the top-level cell if the user is editing in place in this window.

**Return Values**

Returns a pointer to the top-level cell of the specified layout window. If the window is not a layout window, then **NULL** is returned.

**Example**

**Parameters**

- **pWindow**: Pointer to an L-Edit window.

**Version**

Available in L-Edit 10.1 and later versions.

**See Also**

LWindow_GetParameters

```c
void LWindow_GetParameters(
    void** phAppInst,
    void** phParentWnd,
    void** phUserDll
)
```

**Description**

Retrieves the parameters of the L-Edit application window. These parameters can be used with the Windows API. This call is only supported for compiled DLL macros.

In the example below, the `UPI_Entry_Point` function gets the L-Edit application window parameters. These parameters are used by MainFunction to interface with the Windows API.

**Parameters**

- `HINSTANCE *phAppInst` Pointer to application instance.
- `HWND *phParentWnd` Pointer to parent window handle.
- `HINSTANCE *phUserDll` Pointer to DLL handle.

**Example**

```c
HINSTANCE hInst = NULL;
HWND hWnd = NULL;
HINSTANCE hLib = NULL;
LWindow_GetParameters(( void**)&hInst, (void**)&hWnd, (void**)&hLib);

char szModuleName[255];
GetModuleFileName( (HMODULE)hLib, szModuleName, sizeof(szModuleName) );

// More Processing
// ...
```

**See Also**

“Interface Functions” (page 1075), “Windows Functions” (page 1128)
LWindow_GetWindowHandle

```c
void* LWindow_GetWindowHandle( LWindow pWindow )
```

**Description**

Retrieves the window handle to an L-Edit window, creating an interface to the Windows API. This call is only supported for compiled DLL macros.

**Return Values**

Returns a windows handle, cast as a void pointer, for the window pointed to by `pWindow`. To use the window handle, it must be cast to a `HWND` data type, as shown in the example.

**Parameters**

- `pWindow` Pointer to an L-Edit window.

**Example**

```c
LWindow pWindow = LWindow_GetVisible();
if( Assigned( pWindow ) )
{
    void* pvHWnd = LWindow_GetWindowHandle( pWindow );

    // Convert the Windows Handle from a void pointer to a HWND
    HWND phWnd = (HWND) (pvHWnd);

    // Use the Windows API to get the text in the title bar
    char szTitleText[256];
    GetWindowText( phWnd, szTitleText, sizeof(szTitleText) );

    // More processing
    // ...
}
```

**Version**

Available in L-Edit 9.0 and later versions.

**See Also**

“Interface Functions” (page 1075), “Windows Functions” (page 1128), “LWindow” (page 1804)
**LWindow_NewTextWindow**

```c
LWindow LWindow_NewTextWindow( const char* cszFileName,
    LWindowType eWindowType );
```

**Description**

Creates a new text window in L-Edit of the type `eWindowType`. The type indicates the file type for syntax highlighting. If `szFileName` is `NULL`, then a unique filename will be created.

**Return Values**

Returns the `LWindow` pointer to the new window. Returns `NULL` if `eWindowType` is not `TEXT`, `CODE`, `LW_SPICE`, or `LW_LOG` type.

**Parameters**

- `szFileName` File name including path of the new text file. If `szFileName` is `NULL`, then a unique filename will be created.
- `eWindowType` Window type to indicate the file type for syntax highlighting.

**Example**

```c
LCell pCell = LCell_GetVisible();
LFile pTDBFile = LCell_GetFile( pCell );
if( Assigned( pCell ) && Assigned( pTDBFile ) )
{
    char szCellName[MAX_CELL_NAME];
    LCell_GetName( pCell, szCellName, sizeof(szCellName) );

    char szTDBFileName[MAX_TDBFILE_NAME];
    LFile_GetName( pTDBFile, szTDBFileName, sizeof( szTDBFileName ) );

    const char* pszText = LFormat("Active Cellname = \\
            %s\\nActive TDB Filename = \\
            %s\n", szCellName, szTDBFileName );

    LWindow pWindow = LWindow_NewTextWindow( NULL, TEXT );
    LWindowSetText( pWindow, pszText );
    if( LWindow_SaveToFile( pWindow, NULL ) == LStatusOK )
    {
        LWindow_Close( pWindow );
    }
}
```

**Version**

Available in L-Edit 10.0 and later versions.

**See Also**

LWindow_LoadTextFile

LWindow LWindow_LoadTextFile( const char* cszFileName, LWindowType eWindowType );

Description

Loads a text file into a new text window in L-Edit. The type indicates the file type for syntax highlighting.

Return Values

Returns the LWindow pointer to the new window. If an error occurs, NULL is returned due to one of the following possible errors:

- szFileName is NULL.
- eWindowType is not TEXT, CODE, LW_SPICE, or LW_LOG type.

Parameters

**szFileName** Name of the file to load including path.

**eWindowType** Window type to indicate the file type for syntax highlighting.

Example

```c
char szFileName[] = "C:\\MyData\\Results.dat";
LWindow pWindow = LWindow_LoadTextFile( szFileName, TEXT );
if( LWindow_MakeVisible( pWindow ) == LStatusOK )
{
    // More Processing
    // ...
}
```

Version

Available in L-Edit 10.0 and later versions.

See Also

"Interface Functions" (page 1075), "Windows Functions" (page 1128), "LWindow" (page 1804), "LWindowType" (page 1805)
LWindow_SaveToFile

```c
LStatus LWindow_SaveToFile( LWindow pWindow, const char* cszFileName );
```

**Description**

Saves an L-Edit text window to a disk file. The type indicates the file type for syntax highlighting.

**Return Values**

- **LStatusOK** if successful. If an error occurs, **LStatus** contains the error type with possible values:
  - **LBadParameter** - `pWindow` is `NULL`.
  - **LBadParameter** - `szFileName` is `NULL`.

**Parameters**

- `szFileName` Name of the file to load including path.
- `eWindowType` Window type to indicate the file type for syntax highlighting.

**Example**

```c
LCell pCell = LCell_GetVisible();
LFile pTDBFile = LCell_GetFile( pCell );
if( Assigned( pCell ) && Assigned( pTDBFile ) )
{
    char szCellName[MAX_CELL_NAME];
    LCell_GetName( pCell, szCellName, sizeof( szCellName ) );

    char szTDBFileName[MAX_TDBFILE_NAME];
    LFile_GetName( pTDBFile, szTDBFileName, sizeof( szTDBFileName ) );

    const char* pszText = LFormat( "Active Cellname = \"%s\".\nActive TDB Filename = \"%s\"." , szCellName, szTDBFileName );

    LWindow pWindow = LWindow_NewTextWindow( NULL, TEXT );
    LWindow_SetText( pWindow, pszText );
    if( LWindow_SaveToFile( pWindow, NULL ) == LStatusOK )
        LWindow_Close( pWindow );
}
```

**Version**

Available in L-Edit 10.0 and later versions.

**See Also**

LWindow_GetText

unsigned int LWindow_GetText( LWindow pWindow, char* szBuffer, unsigned int nBufferLength)

Description

Retreives the contents of a text window.

Return Values

- If buffer is large enough, then the number of bytes copied to buffer, including terminating zero, is returned.
- If the buffer is too small, then the size of needed buffer is returned.
- If pWindow is NULL or it is not a text window, then 0 is returned.

Parameters

pWindow Pointer to an L-Edit window.
szBuffer Buffer string to store the context of the text window in.
nBufferLength Size of the buffer.

Example

LWindow pWindow = LWindow_GetVisible();
if( Assigned( pWindow ) )
{
    char* pszText = NULL;
    unsigned int nTextSize = LWindow_GetText( pWindow, pszText, 0 );
    pszText = (char*)malloc( nTextSize*sizeof(char) );
    LWindow_GetText( pWindow, pszText, nTextSize );
    // More Processing
    // ...
    free( pszText );
}

Version

Available in L-Edit 10.0 and later versions.

See Also

“Interface Functions” (page 1075), “Windows Functions” (page 1128), “LWindow” (page 1804)
**LWindow_SetText**

```c
LStatus LWindow_SetText( LWindow pWindow, const char* cszText )
```

**Description**

Sets the contents of a text window.

**Return Values**

- **LStatusOK** if successful. If an error occurs, **LStatus** contains the error type with possible values:
  - **LBadParameter** - `pWindow` is **NULL**.
  - **LBadParameter** - `szText` is **NULL**.
  - **LBadParameter** - `pWindow` is not a text window.

**Parameters**

- `pWindow`  
  Pointer to an L-Edit window.
- `szText`  
  Text to put in the L-Edit text window.

**Example**

```c
LCell pCell = LCell_GetVisible();
LFile pTDBFile = LCell_GetFile( pCell );
if( Assigned( pCell ) && Assigned( pTDBFile ) )
{
    char szCellName[MAX_CELL_NAME];
    LCell_GetName( pCell, szCellName, sizeof( szCellName ) );

    char szTDBFileName[MAX_TDBFILE_NAME];
    LFile_GetName( pTDBFile, szTDBFileName, sizeof(szTDBFileName) );

    const char* pszText = LFormat( "Active Cellname = \"%s\".\nActive TDB Filename = \"%s\", szCellName, szTDBFileName );

    LWindow pWindow = LWindow_NewTextWindow( NULL, TEXT );
    LWindow_SetText( pWindow, pszText );
    if(LWindow_SaveToFile( pWindow, NULL ) == LStatusOK )
        LWindow_Close( pWindow );
}
```

**Version**

Available in L-Edit 10.0 and later versions.

**See Also**

- "**Interface Functions**" (page 1075), "**Windows Functions**" (page 1128), "**LStatus**" (page 1792), "**LWindow**" (page 1804)
LWindow_GetName

```c
unsigned int LWindow_GetName( LWindow pWindow, char* szFileName, unsigned int nBufferLength );
```

Description

Retrieves the filename of a text window.

Return Values

- If buffer is large enough, then the number of bytes copied to buffer, including terminating zero, is returned.
- If the buffer is too small, then the size of needed buffer is returned.
- If `pWindow` is `NULL` or it is not a text window, then 0 is returned.

Parameters

- `pWindow` Pointer to an L-Edit window.
- `szFileName` Buffer string to store the filename.
- `nBufferLength` Size of the buffer.

Example

Version

Available in L-Edit 10.1 and later versions.

See Also

- “Interface Functions” (page 1075), “Windows Functions” (page 1128), “LWindow_SetName” (page 1152), “LWindow” (page 1804)
LWindow_SetName

LStatus LWindow_SetName( LWindow pWindow, const char* cszFileName );

Description

Sets the filename of a text window.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

- LBadParameter - pWindow is NULL.
- LBadParameter - szFileName is NULL.
- LBadParameter - pWindow is not a text window.

Parameters

pWindow Pointer to an L-Edit window.

szFileName Filename for the text window.

Example

Version

Available in L-Edit 10.1 and later versions.

See Also

Database Functions

Database functions allow you to create and manipulate a design database. These functions are grouped in the following categories:

- “Application Functions” (page 1154)
- “File Functions” (page 1193)
- “Cell Functions” (page 1260)
- “Instance Functions” (page 1324)
- “Entity Functions” (page 1345)
- “Object Functions” (page 1371)
- “Selection Functions” (page 1494)
- “Layer Functions” (page 1528)
- “Technology Setup Functions” (page 1590)
- “Import/Export Functions” (page 1608)
- “DRC Functions” (page 1622)
- “Extract Functions” (page 1643)
- “Core Functions” (page 1656)
- “Utility Functions” (page 1662)
Application Functions

“LApp_GetVersion” (page 1165)
“LApp_GetVersionDateTime” (page 1166)
“LApp_GetCacheInstances” (page 1155)
“LApp_GetCacheInstancesSmallerThanNumOfPixels” (page 1156)
“LApp_GetFillObjectsDuringDrawing” (page 1157)
“LApp_GetInterruptableRendering” (page 1163)
“LApp_GetRedrawAllWindows” (page 1164)
“LApp_GetHideSmallObjects” (page 1162)
“LApp_GetHideObjectsSmallerThanNumOfPixels” (page 1160)
“LApp_GetHideInstanceInsidesIfLessThanNumOfPixels” (page 1158)
“LApp_GetHideSmallInstanceInsides” (page 1161)
“LApp_GetShowDesignWhileRendering” (page 1168)
“LApp_GetShowDesignFirstTimeIncrement” (page 1169)
“LApp_GetShowDesignNextTimeIncrement” (page 1170)
“LApp_GetRenderingUseCPUForColorMixing” (page 1171)
“LApp_GetRenderingUseMMX” (page 1172)
“LApp_GetRenderingUsePatBltForPatterns” (page 1173)
“LApp_GetAllowSelectionOnLockedLayers” (page 1174)
“LApp_GetFullVersion” (page 1167)
“LApp_SetExportMaskDataExportHiddenObjects” (page 1177)
“LApp_SetCacheInstances” (page 1175)
“LApp_SetCacheInstancesSmallerThanNumOfPixels” (page 1176)
“LApp_SetFillObjectsDuringDrawing” (page 1178)
“LApp_SetInterruptableRendering” (page 1184)
“LApp_SetRedrawAllWindows” (page 1185)
“LApp_SetHideSmallObjects” (page 1183)
“LApp_SetHideObjectsSmallerThanNumOfPixels” (page 1181)
“LApp_SetHideInstanceInsidesIfLessThanNumOfPixels” (page 1179)
“LApp_SetHideSmallInstanceInsides” (page 1182)
“LApp_SetShowDesignWhileRendering” (page 1186)
“LApp_SetShowDesignFirstTimeIncrement” (page 1187)
“LApp_SetShowDesignTimeIncrement” (page 1187)
“LApp_SetRenderingUseCPUForColorMixing” (page 1188)
“LApp_SetRenderingUseMMX” (page 1189)
“LApp_SetRenderingUsePatBltForPatterns” (page 1190)
“LApp_SetAllowSelectionOnLockedLayers” (page 1191)
“LApp_ExitAfterCompletion” (page 1192)
**LApp_GetCacheInstances**

```c
LBoolean LApp_GetCacheInstances( void );
```

**Description**

Retrieves Cache Instances flag.

**Return Values**

`LTRUE` or `LFALSE` depending on Cache Instances flag.

**Example**

```c
LBoolean bCacheInst = LApp_GetCacheInstances();
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LApp_GetCacheInstancesSmallerThanNumOfPixels” (page 1156)
LApp_GetCacheInstancesSmallerThanNumOfPixels

    int LApp_GetCacheInstancesSmallerThanNumOfPixels( void );

Description

Retrieves maximum size of cached instances, in pixels.

Return Values

Returns the maximum size, in pixels, of cached instances.

Example

    int nCacheInstSize = LApp_GetCacheInstancesSmallerThanNumOfPixels();

Version

Available in L-Edit 8.4 and later versions.

See Also

“LApp_GetCacheInstances” (page 1155)
LApp_GetFillObjectsDuringDrawing

LBoolean LApp_GetFillObjectsDuringDrawing( void );

Description

Retrieves Fill Objects During Drawing flag.

Return Values

LTRUE or LFALSE depending on Fill Objects During Drawing flag.

Example

LBoolean bFODD = LApp_GetFillObjectsDuringDrawing();

Version

Available in L-Edit 8.4 and later versions.

See Also

“LApp_SetFillObjectsDuringDrawing” (page 1178)
LApp_GetHideInstanceInsidesIfLessThanNumOfPixels

LStatus LApp_GetHideInstanceInsidesIfLessThanNumOfPixels( LPoint *pptPixelSize );

Description

Gets the pixel size for hiding instance inside. This function stores the results in pptPixelSize. If an instance is smaller than pptPixelSize then its insides are hidden. This gets the Hide instance insides if less than—Horizontal and Vertical parameter on the Setup Application > Rendering tab.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

LBadParameter—pptPixelSize is NULL.

Parameters

pptPixelSize Horizontal number of pixels and vertical number of pixels.

Example

// Save the setting.
LPoint ptCurrentPixelSize;
if( LApp_GetHideInstanceInsidesIfLessThanNumOfPixels( 
    &ptCurrentPixelSize ) == LStatusOK )
{
    LPoint ptNewPixelSize;
    ptNewPixelSize.x = 5;
    ptNewPixelSize.y = 5;
    if( LApp_SetHideInstanceInsidesIfLessThanNumOfPixels( 
        &ptNewPixelSize ) == LStatusOK )
    {
        // More Processing
        // ...
        // Reset the setting.
        LApp_SetHideInstanceInsidesIfLessThanNumOfPixels(&ptCurrentPixelSize);
    }
}

Version

Available in L-Edit 8.3 and later versions.
See Also

LApp_GetHideObjectsSmallerThanNumOfPixels

    int LApp_GetHideObjectsSmallerThanNumOfPixels( void );

Description

Gets the pixel size for hiding objects. If an object is smaller than the returned number of pixels then it is
not rendered. This gets the *Hide objects smaller than--Pixels* parameter on the **Setup Application > Rendering** tab.

Return Values

The number of pixels below which objects will be hidden.

Example

    // Save the setting.
    int iCurrentPixelSize = LApp_GetHideObjectsSmallerThanNumOfPixels();

    if( LApp_SetHideObjectsSmallerThanNumOfPixels( 5 ) == LStatusOK )
    {
        // More Processing
        // ...

        // Reset the setting.
        LApp_SetHideObjectsSmallerThanNumOfPixels( iCurrentPixelSize );
    }

Version

Available in L-Edit 8.3 and later versions.

See Also

“Application Functions” on page 1154, “LStatus” (page 1792),
“LApp_GetHideSmallObjects” (page 1162), “LApp_SetHideSmallObjects” (page 1183),
“LApp_SetHideObjectsSmallerThanNumOfPixels” (page 1181).
**LApp_GetHideSmallInstanceInsides**

```c
LBoolean LApp_GetHideSmallInstanceInsides( void );
```

**Description**

Gets the flag indicating whether to hide small instance insides. This gets the Hide instance insides if less than application setting on the **Setup Application > Rendering** tab.

**Return Values**

LTRUE if the instance insides less than a certain pixel size are hidden, LFALSE to show them.

**Example**

```c
// Save the setting.
LBoolean bHideInsides = LApp_GetHideSmallInstanceInsides();

LApp_SetHideSmallInstanceInsides( LTRUE );
// More Processing
// ...

// Reset the setting.
LApp_SetHideSmallInstanceInsides( bHideInsides );
```

**Version**

Available in L-Edit 8.3 and later versions.

**See Also**

LApp_GetHideSmallObjects

LApp_GetHideSmallObjects( void );

Description

Gets the flag indicating whether to hide small objects. This gets the Hide objects smaller than application setting on the Setup Application > Rendering tab.

Return Values

LTRUE if the objects smaller than a certain pixel size are hidden, LFALSE to show them.

Example

// Save the setting.
LBoolean bHideObjects = LApp_GetHideSmallObjects();
LApp_SetHideSmallObjects( LTRUE );
// More Processing
// ...
// Reset the setting.
LApp_SetHideSmallObjects( bHideObjects );

Version

Available in L-Edit 8.3 and later versions.

See Also

**LApp_GetInterruptableRendering**

```c
LBoolean LApp_GetInterruptableRendering( void );
```

**Description**

Retrieves Interruptable Rendering flag.

**Return Values**

*LTRUE* or *LFALSE* depending on Interruptable Rendering flag.

**Example**

```c
LBoolean bInterruptable = LApp_GetInterruptableRendering();
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LApp_SetInterruptableRendering” (page 1184)
LApp_GetRedrawAllWindows

LBoolean LApp_GetRedrawAllWindows( void );

Description

Retrieves Redraw All Windows flag.

Return Values

LTRUE or LFALSE depending on Redraw All Windows flag.

Example

LBoolean bRedrawAll = LApp_GetRedrawAllWindows();

Version

Available in L-Edit 8.4 and later versions.

See Also

“LApp_SetRedrawAllWindows” (page 1185)
LApp_GetVersion

LStatus LApp_GetVersion( char *VersionString, int BufferLen );

Description

Writes the current L-Edit version number to a string.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadParameters</td>
<td>VersionString is NULL.</td>
</tr>
<tr>
<td>LBufferTooSmall</td>
<td>The buffer size specified by BufferLen is not large enough to store the version string.</td>
</tr>
</tbody>
</table>

Parameters

- **VersionString**: The string buffer in which to store the version number.
- **BufferLen**: Size of the VersionString string buffer.

Example

/* This example displays the current L-Edit version and date/time information in a message box */

char Version[256];
char VersionDateTime[256];
if( (LApp_GetVersion(Version, 255) == LStatusOK) && (LApp_GetVersionDateTime(VersionDateTime, 255) == LStatusOK) )
{
    char LEditFullVersion[600];
    sprintf( LEditFullVersion, "%s %s", Version, VersionDateTime );
    LDialog_MsgBox( LEditFullVersion );
}

Version

Available in L-Edit 8.4 and later versions.

See Also

“LApp_GetVersionDateTime” (page 1166), “LStatus” (page 1792)
LApp_GetVersionDateTime

LStatus LApp_GetVersionDateTime( char *VersionString, int BufferLen );

Description

Writes the date and time of the current L-Edit version to the specified string.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadParameters</td>
<td>VersionString is NULL.</td>
</tr>
<tr>
<td>LBufferTooSmall</td>
<td>The buffer size specified by BufferLen is not large enough to store the version string.</td>
</tr>
</tbody>
</table>

Parameters

VersionString  The string buffer in which to store the version date and time.
BufferLen  Size of the VersionString string buffer.

Example

/* This example displays the current L-Edit version and date/time information in a message box */

char szVer[256];
char szDate[256];
if( ( LApp_GetVersion( szVer, sizeof(szVer) ) == LStatusOK ) &&
    ( LApp_GetVersionDateTime( szDate, sizeof( szDate ) ) == LStatusOK ) )
{
    char LEditFullVersion[600];
    sprintf( LEditFullVersion, "L-Edit %s %s", szVer, szDate );
    LDialog_MsgBox( LEditFullVersion );
}

Version

Available in L-Edit 8.4 and later versions.

See Also

“LApp_GetVersion” (page 1165), “LStatus” (page 1792)
LApp_GetFullVersion

LStatus LApp_GetFullVersion( char* szVersionString, int nBufferLen );

Description
LApp_GetShowDesignWhileRendering

LBoolean LApp_GetShowDesignWhileRendering( void );
LApp_GetShowDesignFirstTimeIncrement

    int LApp_GetShowDesignFirstTimeIncrement( void );
LApp_GetShowDesignNextTimeIncrement

    int LApp_GetShowDesignNextTimeIncrement( void );
LApp_GetRenderingUseCPUForColorMixing

LBoolean LApp_GetRenderingUseCPUForColorMixing( void );
LApp_GetRenderingUseMMX

LBoolean LApp_GetRenderingUseMMX( void );
LApp_GetRenderingUsePatBltForPatterns

LBoolean LApp_GetRenderingUsePatBltForPatterns( void );
LApp_GetAllowSelectionOnLockedLayers

LBoolean LApp_GetAllowSelectionOnLockedLayers( void );
LApp_SetCacheInstances

LStatus LApp_SetCacheInstances( LBoolean bCacheInstances );

Description

Sets Cache Instances flag.

Return Values

Returns LStatusOK if successful.

Parameters

bCacheInstances LTRUE or LFALSE

Example

LApp_SetCacheInstances(LFALSE);

Version

Available in L-Edit 8.4 and later versions.

See Also

“LApp_GetCacheInstances” (page 1155),
“LApp_GetCacheInstancesSmallerThanNumOfPixels” (page 1156),
“LApp_SetCacheInstancesSmallerThanNumOfPixels” (page 1176)


### LApp_SetCacheInstancesSmallerThanNumOfPixels

```c
LStatus LApp_SetCacheInstancesSmallerThanNumOfPixels( int iPixelSize );
```

**Description**

Sets the maximum number of pixels at which an instance will be cached.

**Return Values**

Returns `LStatusOK` if successful.

**Parameters**

- `iPixelSize` The number of pixels at which an instance becomes cached.

**Example**

```c
LApp_SetCacheInstancesSmallerThanNumOfPixels( 10 );
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

- “LApp_GetCachelInstances” (page 1155), “LApp_SetCacheInstances” (page 1175),
- “LApp_GetCacheInstancesSmallerThanNumOfPixels” (page 1156)
LApp_SetExportMaskDataExportHiddenObjects

LStatus LApp_SetExportMaskDataIgnoreHiddenObjects(LBoolean bOption)

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value with possible values:

LBadParameters—is NULL.

Version

Available in L-Edit 8.2 and later versions.

See Also

“LStatus” (page 1792).
LApp_SetFillObjectsDuringDrawing

LStatus LApp_SetFillObjectsDuringDrawing( LBoolean bFillObjectDuringDrawing );

Description

Sets Fill Objects During Drawing flag.

Return Values

Returns \texttt{LStatusOK} if successful.

Parameters

\texttt{bFillObjectDuringDrawing} \quad \texttt{LTRUE} or \texttt{LFALSE}

Example

LApp_SetFillObjectDuringDrawing( LFALSE );

Version

Available in L-Edit 8.4 and later versions.

See Also

“\texttt{LApp_GetFillObjectsDuringDrawing}” (page 1157)
LApp_SetHideInstanceInsidesIfLessThanNumOfPixels

```c
LStatus LApp_SetHideInstanceInsidesIfLessThanNumOfPixels(
    const LPoint *pptPixelSize);
```

**Description**

Sets the pixel size for hiding instance inside. If an instance is smaller than `pptPixelSize` then its insides are hidden. This sets the Hide instance insides if less than—Horizontal and Vertical parameter on the Setup Application > Rendering tab.

**Return Values**

- `LStatusOK` if successful. If an error occurs, `LStatus` contains the error type with possible values:
  - `LBadParameter`—`pptPixelSize` is NULL, `pptPixelSize->x < 0`, `pptPixelSize->y < 0`, `pptPixelSize->x > 32000`, or `pptPixelSize->y > 32000`.

**Parameters**

- `pptPixelSize` — Horizontal number of pixels and vertical number of pixels.

**Example**

```c
// Save the setting.
LPoint ptCurrentPixelSize;
if( LApp_GetHideInstanceInsidesIfLessThanNumOfPixels(
    &ptCurrentPixelSize ) == LStatusOK )
{
    LPoint ptNewPixelSize;
    ptNewPixelSize.x = 5;
    ptNewPixelSize.y = 5;
    if( LApp_SetHideInstanceInsidesIfLessThanNumOfPixels(
        &ptNewPixelSize) == LStatusOK )
    {
        // More Processing
        // ...
        // Reset the setting.
        LApp_SetHideInstanceInsidesIfLessThanNumOfPixels(&ptCurrentPixelSize);
    }
}
```

**Version**

Available in L-Edit 8.3 and later versions.
See Also

“Application Functions” on page 1154, “LStatus” (page 1792), “LBoolean” (page 1697),
“LApp_GetHideInstanceInsidesIfLessThanNumOfPixels” (page 1158),
“LApp_GetHideSmallInstanceInsides” (page 1161),
“LApp_SetHideSmallInstanceInsides” (page 1182),
“LApp_SetHideInstanceInsidesIfLessThanNumOfPixels” (page 1179).
LApp_SetHideObjectsSmallerThanNumOfPixels

LStatus LApp_SetHideObjectsSmallerThanNumOfPixels( int iPixelSize );

Description

Sets the pixel size for hiding objects. If an object is smaller than $iPixelSize$ then it is not rendered. This sets the Hide objects smaller than--Pixels parameter on the Setup Application > Rendering tab.

Return Values

$LStatusOK$ if successful. If an error occurs, $LStatus$ contains the error type with possible values:

$LBadParameter$ — $iPixelSize < 0$ or $iPixelSize > 32000$.

Parameters

$iPixelSize$ The number of pixels below which objects will be hidden.

Example

// Save the setting.
int iCurrentPixelSize = LApp_GetHideObjectsSmallerThanNumOfPixels();

if( LApp_SetHideObjectsSmallerThanNumOfPixels( 5 ) == LStatusOK )
{
    // More Processing
    // ...

    // Reset the setting.
    LApp_SetHideObjectsSmallerThanNumOfPixels( iCurrentPixelSize );
}

Version

Available in L-Edit 8.3 and later versions.

See Also

**LApp_SetHideSmallInstanceInsides**

```c
LStatus LApp_SetHideSmallInstanceInsides( LBoolean bHideSmallInstanceInsides );
```

**Description**

Sets the flag indicating whether to hide small instance insides. This sets the `Hide instance insides if less than--Pixels` application parameter on the **Setup Application > Rendering** tab.

**Return Values**

Always `LStatusOK`.

**Parameters**

- `bHideSmallInstanceInsides` - `LTRUE` to hide instance insides, `LFALSE` to show instance insides.

**Example**

```c
// Save the setting.
LBoolean bHideInsides = LApp_GetHideSmallInstanceInsides();
LApp_SetHideSmallInstanceInsides(LTRUE);
// More Processing
// ...
// Reset the setting.
LApp_SetHideSmallInstanceInsides( bHideInsides );
```

**Version**

Available in L-Edit 8.3 and later versions.

**See Also**

LApp_SetHideSmallObjects

LStatus LApp_SetHideSmallObjects(LBoolean bHideSmallObjects);

Description

Sets the flag indicating whether to hide small objects. This sets the Hide objects smaller than application parameter on the Setup Application > Rendering tab.

Return Values

Always LStatusOK.

Parameters

bHideSmallObjects  LTRUE if the objects smaller than a certain pixel size are hidden, LFALSE to show them.

Example

// Save the setting.
LBoolean bHideObjects = LApp_GetHideSmallObjects();

LApp_SetHideSmallObjects( LTRUE );
// More Processing
// ...

// Reset the setting.
LApp_SetHideSmallObjects( bHideObjects );

Version

Available in L-Edit 8.3 and later versions.

See Also

LApp_SetInterruptableRendering

LStatus LApp_SetInterruptableRendering( LBoolean bInterruptRendering );

Description

Sets the Interruptable Rendering flag.

Return Values

Always LStatusOK.

Parameters

bInterruptRendering LTRUE if rendering can be interrupted, LFALSE if it cannot.

Example

LApp_SetInterruptableRendering( LFALSE );

Version

Available in L-Edit 8.4 and later versions.

See Also

“LApp_GetInterruptableRendering” (page 1163).
LApp_SetRedrawAllWindows

```c
LStatus LApp_SetRedrawAllWindows( LBoolean bRedrawAllWindows );
```

**Description**

Sets the Redraw All Windows flag.

**Return Values**

Always `LStatusOK`.

**Parameters**

- `bRedrawAllWindows` 
  - `LTRUE` if windows should be redrawn, `LFALSE` if not.

**Example**

```c
LApp_SetRedrawAllWindows( LFALSE );
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LApp_GetRedrawAllWindows” (page 1164).
LApp_SetShowDesignWhileRendering

LStatus LApp_SetShowDesignWhileRendering( LBoolean bShowDesignWhileRendering );
LApp_SetShowDesignTimeIncrement

LStatus LApp_SetShowDesignTimeIncrement( int nFirstIncrement, int nNextIncrement );
LApp_SetRenderingUseCPUForColorMixing

LStatus LApp_SetRenderingUseCPUForColorMixing( LBoolean bUseCPUForColorMixing );
LApp_SetRenderingUseMMX

LStatus LApp_SetRenderingUseMMX( LBoolean bUseMMX );
LApp_SetRenderingUsePatBltForPatterns

```c
LStatus LApp_SetRenderingUsePatBltForPatterns( LBoolean bUsePatBltForPatterns );
```
LApp_SetAllowSelectionOnLockedLayers

LStatus LApp_SetAllowSelectionOnLockedLayers( LBoolean bInterruptRendering );
LApp_ExitAfterCompletion

```c
void LApp_ExitAfterCompletion( bool bDiscardUnsavedFiles );
```

Description

Calling `LApp_ExitAfterCompletion` will cause L-Edit to exit upon the completion of the macro. Make sure to make this call after all macros called from within the current one are completed. If `bDiscardUnsavedFiles` is not set and any open documents are not saved, L-Edit will not exit.

Parameters

- **bDiscardUnsavedFiles**
  - If `bDiscardUnsavedFiles` is true L-Edit will exit regardless of whether any unsaved files (TDB, text, or other) are open. If `bDiscardUnsavedFiles` is false L-Edit will exit only if there are no open unsaved documents.

Example

```c
void MyMacro
{
    //perform macro-specific operations
    ...

    //make sure all documents that need to be saved are saved
    ...

    //signal L-Edit to terminate upon completion of MyMacro
    LApp_ExitAfterCompletion( true );
}
```

Version

Available in L-Edit 8.42 and later versions.
File Functions

TDB (Tanner Database) files are design files in a Tanner Research proprietary format. A TDB file is the highest level of the L-Edit database hierarchy. It is composed of linearly linked lists of cells and layers.

A TDB file is the highest level of the L-Edit database hierarchy. A single TDB file usually contains the complete design for a chip or MCM, but it may also consist of a library of cells or a partial design to be merged with other design files.

The file functions below allow the user to manipulate an L-Edit design file.

“LFile_New” (page 1195)
“LFile_Open” (page 1196)
“LFile_OpenCell” (page 1197)
“LFile_Save” (page 1198)
“LFile_Close” (page 1200)
“LFile_GetList” (page 1202)
“LFile_GetLock” (page 1204)
“LFile_IsChanged” (page 1206)
“LFile.GetName” (page 1208)
“LFile_GetAuthor” (page 1209)
“LFile_GetFabricationCell” (page 1211)
“LFile_GetOrganization” (page 1213)
“LFile_GetLayoutVersion” (page 1215)
“LFile_GetSetupVersion” (page 1217)
“LFile_GetInfoText” (page 1219)
“LFile_GetEnvironment” (page 1221)
“LFile_GetGrid” (page 1223)
“LFile_GetGrid_v10_00” (page 1225)
“LFile_SetGridEx840” (page 1229)
“LFile_SetCurveSetup” (page 1231)
“LFile_SetSelectionParam” (page 1235)
“LFile_DeleteUserData” (page 1238)
“LFile_DisplayCellBrowser” (page 1240)
“LFile_GetVisible” (page 1246)
“LFile_GetGrid_v10_00” (page 1225)
“LFile_GetDisplayUnitInfo” (page 1249)
“LFile_IntUtoDispU” (page 1251)
“LFile_IntUtoMicrons” (page 1253)
“LFile_SaveAs” (page 1199)
“LFile_Find” (page 1201)
“LFile_GetNext” (page 1203)
“LFile_SetLock” (page 1205)
“LFile_SetChanged” (page 1207)
“LFile_SetAuthor” (page 1210)
“LFile_SetFabricationCell” (page 1212)
“LFile_SetOrganization” (page 1214)
“LFile_SetLayoutVersion” (page 1216)
“LFile_SetSetupVersion” (page 1218)
“LFile_SetInfoText” (page 1220)
“LFile_SetEnvironment” (page 1222)
“LFile_GetGridEx840” (page 1224)
“LFile_SetGrid_v10_00” (page 1226)
“LFile_GetCurveSetup” (page 1231)
“LFile_GetSelectionParam” (page 1234)
“LFile_GetUserData” (page 1236)
“LFile_SetUserData” (page 1237)
“LFile_ClearUserData” (page 1239)
“LFile_GetResolvedFileName” (page 1244)
“LFile_SetLastCurrent” (page 1241)
“LFile_SetGrid_v10_00” (page 1226)
“LFile_SetDisplayUnit” (page 1250)
“LFile_DispuToIntu” (page 1252)
“LFile_MicronsToIntu” (page 1254)
“LFile_GetGrid” (page 1223)
“LFile_GetGridEx840” (page 1224)
“LFile_IntUtoLocU” (page 1247)
“LFile_GetCurveSetup” (page 1231)
“LFile_GetDesignRuleFlags” (page 1242)
“LFile_GetResolvedFileName” (page 1244)
“LFile_IntUtoLocU” (page 1247)
“LFile_GetDisplayUnitInfo” (page 1249)
“LFile_IntUtoDispU” (page 1251)
“LFile_IntUtoMicrons” (page 1253)
“LFile_GetCurrentLayerPalette” (page 1255)
“LFile_GetNextLayerPalette” (page 1257)

“LFile_SetGrid” (page 1228)
“LFile_SetGridEx840” (page 1229)
“LFile_LocUtoIntU” (page 1248)
“LFile_SetCurveSetup” (page 1232)
“LFile_SetDesignRuleFlags” (page 1243)
“LFile_GetVisible” (page 1246)
“LFile_LocUtoIntU” (page 1248)
“LFile_SetDisplayUnit” (page 1250)
“LFile_DispUtoIntU” (page 1252)
“LFile_MicronsToIntU” (page 1254)
“LFile_SetCurrentLayerPalette” (page 1256)
“LFile_DefineLayerPalette” (page 1258)
**LFile_New**

```c
LFile LFile_New( LFile setup_file, char* name );
```

**Description**

Creates a new, empty layout file with a technology setup copied from the specified file.

**Return Values**

Returns a pointer to the new file, or NULL on error.

**Parameters**

- `setup_file`: File whose setup is to be used (if NULL, then the setup of the current file is used).
- `name`: Name of the new file.

**Example**

```c
// make a new file, based on current file
LFile pCurFile = LFile_GetVisible();
if ( pCurFile )
{
    LFile pFile = LFile_New( pCurFile, "My new file" );
    if ( ! pFile )
        LDialog_AlertBox( "Failed to create file" );
}
```

**See Also**

“File Functions” (page 1193)
LFile_Open

LFile LFile_Open( const char* name, LFileType type );

Description

Opens a TDB, CIF, or GDS II file.

Return Values

A pointer to the file, or NULL on error.

Parameters

name       Name of the file to open.

name       Format of the file (LTdbFile, LCifFile, or LGdsFile).

description

Example

LFile pFile = LFile_Open( "c:\My data\My file", LTdbFile );
if ( ! pFile )
    LDialog_AlertBox( "File open failed" );

See Also

“LFile” (page 1736), “LFileType” (page 1737), “File Functions” (page 1193)
**LFile_OpenCell**

```c
LWindow LFile_OpenCell( LFile file, char *cell_name );
```

**Description**

Opens a layout window for the specified cell in the specified file.

**Return Values**

Returns a pointer to the newly created window; otherwise NULL.

**Parameters**

- **file**  
  Specified file.
- **cell_name**  
  Name of the specified cell.

**Example**

```c
if ( ! LFile_OpenWindow( LFile_GetVisible(), "my cell" ) )
    LDialog_AlertBox( "Failed to open 'my cell'" );
```

**See Also**

“UPIDrawingToolType” (page 1809), “LFile” (page 1736), “Interface Functions” (page 1075)
LFile_Save

LStatus LFile_Save( LFile file );

Description

Saves the specified file into a TDB file of the same name (with extension .tdb).

Return Values

LStatusOK if successful. If an error occurs LStatus contains the error value.

Parameters

file

Pointer to the file to be saved.

Example

if ( LStatusOK != LFile_Save( LFile_GetVisible() ) )
    LDialog_AlertBox( "Failed to save current file" );

See Also

“LStatus” (page 1792), “LFile_SaveAs” (page 1199), “File Functions” (page 1193)
**LFile_SaveAs**

```c
LStatus LFile_SaveAs( LFile file, const char* name, LFileType type );
```

**Description**

Saves a file as a different file with the specified name and file type.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- **file** File to be saved.
- **name** Name under which the file is to be saved.
- **type** Format in which the file is to be saved (LTdbFile, LCifFile, or LGdsFile).

**Example**

```c
if ( LStatusOK != LFile_SaveAs( LFile_GetVisible(), "newname", LTdbFile))
    LDialog_AlertBox( "Failed to save new copy of current file" );
```

**See Also**

“LStatus” (page 1792), “LFileType” (page 1737), “LFile_Save” (page 1198), “File Functions” (page 1193)
**LFile_Close**

```c
LStatus LFile_Close( LFile file );
```

**Description**

Closes the specified file without checking for changes.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `file` : File to be closed.

**Example**

```c
// NOTE: pending changes are LOST!
if ( LFile_GetVisible() )
    LFile_Close( LFile_GetVisible() );
```

**See Also**

LFile_Find

LFile LFile_Find( const char* name );

Description

Finds a file in a list of open files whose name matches the specified string.

Return Values

Returns a pointer to the file, if found; otherwise returns NULL.

Parameters

name Name (without filename extension) of the file to be searched.

Example

LFile pFile = LFile_Find( "my file" );
if ( !pFile )
    LDialog_AlertBox( "'my file' not open" );

See Also

“LFile” (page 1736), “File Functions” (page 1193)
LFile_GetList

LFile GetList( void )

Description

Gets a list of currently open files.

Return Values

Returns the head of the list of open files. Returns NULL if no files are open.

Example

// count open files
int nCount = 0;
LFile pFile;
for( pFile = LFile_GetList(); pFile; pFile = LFile_GetNext( pFile ) )
{
   // process each file
   nCount++;
}
LDialog_MsgBox( LFormat( "%d files open", nCount ) );

See Also

“File Functions” (page 1193)
LFile_GetNext

LFile LFile_GetNext( LFile file );

Description

Gets the next file in the list of open files after the specified file.

Return Values

Returns a pointer to the next file in the currently opened file list. If no next file exists, it returns a NULL.

Parameters

file Pointer to a file.

Example

/*This example demonstrates a simple way of traversing all the loaded files*/
/*Declare a L-Edit file variable*/
LFile pFile;
/*Get a list of all the currently loaded files and traverse the list*/
for( pFile = LFile_GetList(); pFile; pFile = LFile_GetNext( pFile) )
{
    /*Do processing specific to a file*/
}

See Also

"LFile_GetList" (page 1202), "LCell_GetNext" (page 1268), "LInstance_GetNext" (page 1333),
"File Functions" (page 1193)
**LFile_GetLock**

```c
int LFile_GetLock( LFile file );
```

**Description**

Checks whether a file is locked or not.

**Return Values**

Returns zero if the specified file is unlocked; otherwise returns a nonzero value.

**Parameters**

- `file` File to be checked.

**See Also**

“LFile” (page 1736), “File Functions” (page 1193)
LFile_SetLock

    int LFile_SetLock( LFile file, int set );

Description

Locks or unlocks the specified file. If set is nonzero, the file is locked; if set is zero, the file is unlocked.

Return Values

A nonzero value if the file is locked.

Parameters

    file File to be locked or unlocked.
    set Value that determines the file’s new status: zero unlocks; anything else locks.

See Also

“LFile” (page 1736), “File Functions” (page 1193)
LFile_IsChanged

```c
int LFile_IsChanged( LFile file );
```

Description

Checks the specified file to determine if it has been changed since it was last saved.

Return Values

The function returns 1 if the file has been changed or zero if it has not.

Parameters

`file` File to be checked.

See Also

“LFile” (page 1736), “File Functions” (page 1193)
**LFile_SetChanged**

```c
void LFile_SetChanged( LFile pTDBFile );
```

**Description**

Marks the file as changed. This will also increment the minor version.

**Parameters**

- `pTDBFile` The file to mark as changed.

**Example**

```c
LFile pTDBFile = LFile_GetVisible();
if( Assigned( pTDBFile ) )
{
    // Do some processing on pTDBFile.
    // ...

    // Mark the file as changed.
    LFile_SetChanged( pTDBFile );
}
```

**Version**

Available in L-Edit 8.2 and later versions.

**See Also**

“*File Functions*” (page 1193), “*LFile*” (page 1736), “*LFile_IsChanged*” (page 1206).
LFile_GetName

```c
char* LFile_GetName( LFile file, char* name, const int maxlen );
```

Description

Gets the text of the name of the file including its path but without its extension.

Return Values

Returns a pointer to the string `name`; returns NULL if unsuccessful.

Parameters

- **file**
  - File whose name is to be retrieved.
- **name**
  - String containing the name text (the name buffer).
- **maxlen**
  - Maximum length allowed for `name`.

Example

```c
LFile pFile = LFile_GetVisible();
if ( pFile ) {
    char filename[MAX_TDBFILE_NAME];
    if ( LFile_GetName( pFile, filename, sizeof(filename) ) )
        LDialog_MsgBox( LFormat( "file: %s", filename ) );
}
```

See Also

“LCell_GetName” (page 1271), “LInstance_GetName” (page 1334), “File Functions” (page 1193)
LFile_GetAuthor

char* LFile_GetAuthor( LFile file, char* author, const int maxlen );

Description

Gets the text of the author field in the information summary for the specified file.

Return Values

Returns a pointer to the string author if successful; otherwise returns NULL.

Parameters

file File whose author is to be retrieved.
author String containing the author text.
maxlen Maximum length allowed for author.

Example

LFile pFile = LFile_GetVisible();
if ( pFile )
{
    char name[MAX_TDBFILE_NAME];
    if ( LFile_GetAuthor( pFile, name, sizeof(name) ) )
        LDialog_MsgBox( LFormat("author: %s", name ) );
}

See Also

“LFile” (page 1736), “File Functions” (page 1193)
LFile_SetAuthor

char* LFile_SetAuthor( LFile file, char* author );

Description

Sets the text of the author field in the information summary for the specified file.

Return Values

Returns a pointer to the string author if successful; otherwise NULL.

Parameters

file File whose author text is to be set.
author String containing the author text.

See Also

“LFile” (page 1736), “File Functions” (page 1193)
LFile_GetFabricationCell

```
LCell LFile_GetFabricationCell( LFile file );
```

**Description**

Gets the cell marked as the "top" or "root" cell (the fabrication cell) of the specified file for foundry fabrication.

**Return Values**

Returns a pointer to the fabrication cell if found; otherwise NULL.

**Parameter**

`file` Specified file.

**See Also**

LFile_SetFabricationCell

LStatus LFile_SetFabricationCell( LFile file, LCell cell );

Description

Marks the specified cell as the "top" or "root" cell (the fabrication cell) of the specified file for foundry fabrication, conforming to CIF and GDS II conventions.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file
Specified file.

cell
Cell to be set as the fabrication cell.

See Also

**LFile_GetOrganization**

```c
char* LFile_GetOrganization( LFile file, char* org, const int maxlen );
```

**Description**

Gets the text of the organization field in the information summary for the specified file.

**Return Values**

Returns a pointer to the organization string if successful; otherwise returns NULL.

**Parameters**

- `file`: File whose organization is to be retrieved.
- `org`: String containing the organization text.
- `maxlen`: Maximum length allowed for `org`.

**Example**

```c
LFile pFile = LFile_GetVisible();
if ( pFile )
{
    char name[MAX_TDBFILE_NAME];
    if ( LFile_GetOrganization( pFile, name, sizeof(name) ) )
        LDialog_MsgBox( LFormat("organization: %s", name ) );
}
```

**See Also**

“LFile” (page 1736), “File Functions” (page 1193)
LFile_SetOrganization

const char* LFile_SetOrganization( LFile pTDBFile, const char* szOrg )

Description

Sets the text of the organization field in the information summary for the specified file.

Return Values

Returns a pointer to the file organization buffer if successful; otherwise returns NULL.

Parameters

pTDBFile File whose organization is to be set.
szOrg String containing the organization text.

See Also

“LFile” (page 1736), “File Functions” (page 1193)
LFile_GetLayoutVersion

void LFile_GetLayoutVersion( LFile file, long* major, long* minor );

Description

Gets the major and minor layout version numbers of the specified file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File whose layout version numbers are to be retrieved.</td>
</tr>
<tr>
<td>major</td>
<td>Pointer to the major layout version number.</td>
</tr>
<tr>
<td>minor</td>
<td>Pointer to the minor layout version number.</td>
</tr>
</tbody>
</table>

See Also

“LFile” (page 1736), “File Functions” (page 1193)
LFile_SetLayoutVersion

```c
void LFile_SetLayoutVersion( LFile file, long* major, long* minor );
```

**Description**

Sets the major and minor layout version numbers of the specified file.

**Parameters**

- `file` file whose layout version numbers are to be set.
- `major` Pointer to the major layout version number.
- `minor` Pointer to the minor layout version number.

**See Also**

“LFile” (page 1736), “File Functions” (page 1193)
**LFile_GetSetupVersion**

```c
void LFile_GetSetupVersion( LFile file, long* major, long* minor );
```

**Description**

Gets the major and minor setup numbers of the specified file.

**Parameters**

- `file`  
  File whose setup version numbers are to be retrieved.
- `major`  
  Pointer to the major setup version number.
- `minor`  
  Pointer to the minor setup version number.

**See Also**

“LFile” (page 1736), “File Functions” (page 1193)
LFile_SetSetupVersion

```c
void LFile_SetSetupVersion( LFile file, long* major, long* minor );
```

**Description**

Sets the major and minor setup version numbers of the specified file.

**Return Values**

Returns NULL on error.

**Parameters**

- `file`: File whose setup version numbers are to be set.
- `major`: Pointer to the major setup version number.
- `minor`: Pointer to the minor setup version number.

**See Also**

“LFile” (page 1736), “File Functions” (page 1193)
LFile_GetInfoText

char* LFile_GetInfoText( LFile file, char* info, const int maxlen );

Description

Gets the text of the information field in the file information summary for the specified file.

Return Values

Returns a pointer to the string info if successful; otherwise returns NULL.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File whose information is to be retrieved.</td>
</tr>
<tr>
<td>info</td>
<td>String containing the information text.</td>
</tr>
<tr>
<td>maxlen</td>
<td>Maximum length allowed for info.</td>
</tr>
</tbody>
</table>

Example

LFile pFile = LFile_GetVisible();
if ( pFile )
{
    char name[MAX_TDBFILE_NAME];
    if ( LFile_GetInfoText( pFile, name, sizeof(name) ) )
        LDialog_MsgBox( LFormat("info text: %s", name ) );
}

See Also

“LFile” (page 1736), “File Functions” (page 1193)
LFile_SetInfoText

    char* LFile_SetInfoText( LFile file, char* info );

Description

Sets and returns the text of the information field in the file information summary for the specified file. A NULL value may be given.

Return Values

Returns a pointer to the string info if successful; otherwise returns NULL.

Parameters

file File whose information is to be set.
info String containing the information text.

See Also

“LFile” (page 1736), “File Functions” (page 1193)
LFile_GetEnvironment

LEnvironment *LFile_GetEnvironment( LFile file, LEnvironment *env );

Description

Gets the environment setting of the specified file.

Return Values

Returns a pointer to the structure env if successful; otherwise returns NULL.

Parameters

file

File whose environment setting is to be retrieved.

env

Pointer to the file environment structure.

See Also

“LEnvironment” (page 1730), “LFile” (page 1736), “File Functions” (page 1193)
**LFile_SetEnvironment**

```
LStatus LFile_SetEnvironment( LFile file, LEnvironment *env );
```

**Description**

Sets the environment of the specified file according to the parameters defined in LEnvironment.

**Return Values**

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `file` File whose environment is to be set.
- `env` Pointer to the file environment structure.

**See Also**

“LEnvironment” (page 1730), “LFile” (page 1736), “File Functions” (page 1193)
LFile_GetGrid

LGrid *LFile_GetGrid( LFile file, LGrid *grid );

Description

Gets the grid setting of the specified file.

Note:

Note that this function is superseded by “LFile_GetGridEx840” (page 1224).

Return Values

Returns a pointer to the grid structure if successful; otherwise returns NULL.

Parameters

file File whose grid setting is to be retrieved.

grid Pointer to the grid structure.

See Also

LFile_GetGridEx840

LStatus LFile_GetGridEx840( LFile file, LGridEx840 *grid );

Description

Gets the grid setting of the specified file.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadFile</td>
<td>file is NULL</td>
</tr>
<tr>
<td>LBadParameters</td>
<td>grid is NULL</td>
</tr>
</tbody>
</table>

Parameters

- **file**  
  File whose grid setting is to be retrieved.
- **grid**  
  Pointer to the grid structure.

Example

LGridEx840 Grid;  
LFile_GetGridEx840( MyFile, &Grid );

Version

Available in L-Edit 8.4 and later versions.

**Note:**  
Note that this function is superseded in L-Edit V10 and later.

See Also

“LFile_SetGridEx840” (page 1229), “LGridEx840” (page 1748)
LFile_GetGrid_v10_00

LStatus LFile_GetGrid_v10_00( LFile file, LGrid_v10_00 *grid );

Description

Gets the grid setting of the specified file.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadFile</td>
<td>file is NULL</td>
</tr>
<tr>
<td>LBadParameters</td>
<td>grid is NULL</td>
</tr>
</tbody>
</table>

Parameters

- **file**
  File whose grid setting is to be retrieved.

- **grid**
  Pointer to the grid structure.

See Also

“LGrid_v10_00” (page 1750), “LFile” (page 1736), “File Functions” (page 1193)
LFile_SetGrid_v10_00

LStatus LFile_SetGrid_v10_00( LFile file, LGrid_v10_00 *grid );

Description

Sets the grid information of the specified file.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadFile</td>
<td>file is NULL</td>
</tr>
<tr>
<td>LBadParameters</td>
<td>One or more of the following errors:</td>
</tr>
<tr>
<td></td>
<td>- grid is NULL</td>
</tr>
<tr>
<td></td>
<td>- grid.displayed_grid_size &lt; 1 or grid.displayed_grid_size &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>- grid.min_grid_pixels &gt; 1 or grid.min_grid_pixels &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>- grid.displayed_majorgrid_size &lt; 1 or grid.displayed_majorgrid_size &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>- grid.min_majorgrid_pixels &lt; 1 or grid.min_majorgrid_pixels &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>- grid.mouse_snap_grid_size &lt; 1 or grid.mouse_snap_grid_size &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>- grid.cursor_type is invalid</td>
</tr>
<tr>
<td></td>
<td>- grid.locator_scaling &lt; 1 or grid.locator_scaling &gt; WORLD_MAX</td>
</tr>
</tbody>
</table>

Parameters

file File whose grid is to be set.

grid Pointer to the grid structure.

Example

/* Get the current grid setting for MyFile */
LGrid_v10_00 Grid;
LFile_GetGrid_v10_00( MyFile, &Grid );

/* Specify new grid settings */
Grid.min_majorgrid_pixels = 10 * Grid.min_grid_pixels;
Grid.displayed_majorgrid_size = 10 * Grid.displayed_grid_size;
/* Apply the new grid structure to MyFile */
LFile_SetGrid_v10_00(MyFile, &Grid);

Version

Available in L-Edit 10 and later versions.

See also

“LFile_GetGrid_v10_00” (page 1225), “LGrid_v10_00” (page 1750)
LFile_SetGrid

LStatus LFile_SetGrid( LFile file, LGrid *grid );

Description

Sets the grid of the specified file.

Note: Note that this function is superseded by “LFile_SetGridEx840” (page 1229).

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file File whose grid is to be set.
grid Pointer to the grid structure.

See Also

**LFile_SetGridEx840**

```c
LStatus LFile_SetGridEx840( LFile file, LGridEx840 *grid );
```

**Description**

Sets the grid information of the specified file.

**Return Values**

Returns `LStatusOK` if successful. If an error occurs, `LStatus` contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadFile</td>
<td><code>file</code> is NULL</td>
</tr>
<tr>
<td>LBadParameters</td>
<td>One or more of the following errors:</td>
</tr>
<tr>
<td></td>
<td>- <code>grid</code> is NULL</td>
</tr>
<tr>
<td></td>
<td>- <code>grid.displayed_grid_size</code> &lt; 1 or</td>
</tr>
<tr>
<td></td>
<td><code>grid.displayed_grid_size</code> &gt; WORLD_MAX,</td>
</tr>
<tr>
<td></td>
<td>- <code>grid.min_grid_pixels</code> &gt; 1 or</td>
</tr>
<tr>
<td></td>
<td><code>grid.min_grid_pixels</code> &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>- <code>grid.displayed_majorgrid_size</code> &lt; 1 or</td>
</tr>
<tr>
<td></td>
<td><code>grid.displayed_majorgrid_size</code> &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>- <code>grid.min_majorgrid_pixels</code> &lt; 1 or</td>
</tr>
<tr>
<td></td>
<td><code>grid.min_majorgrid_pixels</code> &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>- <code>grid.mouse_snap_grid_size</code> &lt; 1 or</td>
</tr>
<tr>
<td></td>
<td><code>grid.mouse_snap_grid_size</code> &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>- <code>grid.cursor_type</code> is invalid</td>
</tr>
<tr>
<td></td>
<td>- <code>grid.locator_scaling</code> &lt; 1 or</td>
</tr>
<tr>
<td></td>
<td><code>grid.locator_scaling</code> &gt; WORLD_MAX</td>
</tr>
</tbody>
</table>

**Parameters**

- **file**
  File whose grid is to be set.
- **grid**
  Pointer to the grid structure.

**Example**

```c
/* Get the current grid setting for MyFile */
LGridEx840 Grid;
LFile_GetGridEx840( MyFile, &Grid );

/* Specify new grid settings */
Grid.min_majorgrid_pixels = 10*Grid.min_grid_pixels;
Grid.displayed_majorgrid_size = 10*Grid.displayed_grid_size;
```
/* Apply the new grid structure to MyFile */
LFile_SetGridEx840( MyFile, &Grid );

Version

Available in L-Edit 8.4 and later versions. In L-Edit V10 and later, the curve representation changed to use the manufacturing grid, making this function unnecessary.

Note: Note that this function is superseded in L-Edit V10 and later.

See also

“LFile_GetGridEx840” (page 1224), “LGridEx840” (page 1748)
LFile_GetCurveSetup

LCurve *LFile_GetCurveSetup( LFile file, LCurve *curve );

Description

Gets the curve properties from the specified file and writes them to the destination specified by curve. Curve properties include the maximum number of segments per curve, the maximum segment length, and WYSIWYG display.

Return Values

Returns a pointer to the curve setup properties in case of success; otherwise, returns NULL.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>The specified file.</td>
</tr>
<tr>
<td>curve</td>
<td>Pointer to the curve setup information.</td>
</tr>
</tbody>
</table>

Example

```
LCurve CurveSetup;
LFile_GetCurveSetup(MyFile, &CurveSetup);
```

Version

Available in L-Edit 8.4 and later versions. In L-Edit V10 and later, the curve representation changed to use the manufacturing grid, making this function unnecessary.

See Also

“LFile_SetCurveSetup” (page 1232), “LPolygon_StraightenAllCurves” (page 1451), “LCurve” (page 1710)
**LFile_SetCurveSetup**

```c
LStatus LFile_SetCurveSetup( LFile file, LCurve *curve );
```

**Description**

Sets the curve properties for the specified file. Curve properties include the maximum number of segments per curve, the maximum segment length, and WYSIWYG display.

**Return Values**

Returns `LStatusOK` if successful. If an error occurs, `LStatus` contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadFile</td>
<td><code>file</code> is NULL.</td>
</tr>
<tr>
<td>LBadParameters</td>
<td>Indicates one or more of the following errors:</td>
</tr>
<tr>
<td></td>
<td>- <code>curve</code> is NULL</td>
</tr>
<tr>
<td></td>
<td>- VALID PARAMETER RANGES?</td>
</tr>
</tbody>
</table>

**Parameters**

- `file` The specified file.
- `curve` Pointer to curve setup information.

**Example**

```c
/* Get the curve setup information for MyFile */
LCurve CurveSetup;
LFile_GetCurveSetup(MyFile, &CurveSetup);

/* Edit curve properties */
CurveSetup.mMaxNumSegmentsPerCurve = 100;
CurveSetup.mMaxSegmentLength = 50;

/* Assign the new curve properties to MyFile */
LFile_SetCurveSetup(MyFile, &CurveSetup);

/* Straighten curves on the specified polygon */
LPolygon_StraightenAllCurves(MyCell, MyPolygon);
```

**Version**

Available in L-Edit 8.4 and later versions. In L-Edit V10 and later, the curve representation changed to use the manufacturing grid, making this function unnecessary.
See Also

“LFile_SetCurveSetup” (page 1232), “LPolygon_StraightenAllCurves” (page 1451), “LCurve” (page 1710)
LFile_GetSelectionParam

LSelectionParam *LFile_GetSelectionParam( LFile file, LSelectionParam *param );

Description

Gets the selection parameters of the specified file.

Return Values

Returns a pointer to the selection structure if successful; otherwise returns NULL.

Parameters

file
File whose selection parameter are to be found.

param
Pointer to the selection parameter structure.

See Also

“LWireParam” (page 1808), “File Functions” (page 1193)
**LFile_SetSelectionParam**

```c
LStatus LFile_SetSelectionParam(LFile file, LSelectionParam *param);
```

**Description**

Sets the selection parameters of the given file.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

**Parameters**

- `file` - File whose selection parameters are to be set.
- `param` - Pointer to the selection parameter structure.

**See Also**

**LFile_GetUserData**

```c
void *LFile_GetUserData( LFile file );
```

**Description**

Gets a pointer to user-defined data associated with the specified file.

**Return Values**

Returns a pointer to the user data if successful; otherwise returns NULL.

**Parameter**

`file`  
File whose user-defined data is needed.

**See Also**

“LFile” (page 1736), “File Functions” (page 1193)
**LFile_SetUserData**

```c
LStatus LFile_SetUserData( LFile file, void *dataPointer );
```

**Description**

Uses a data pointer within a file to associate user-defined data with the file. Data can be integer, string, or any other type.

**Return Values**

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameter**

- **file**
  - File which will contain the user-defined data.

- **dataPointer**
  - User-defined data.

**See Also**

LFile_DeleteUserData

LStatus LFile_DeleteUserData( LFile pTDBFile );

Description

Deletes the user-defined expansion pointer in the specified file.

Return Values

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

Parameters

pTDBFile Specified file.

See Also

**LFile_ClearUserData**

```c
LStatus LFile_ClearUserData( LFile TDBFile );
```

**Description**

Sets the user-defined data pointer in the specified TDB file to NULL without freeing it.

**Return Values**

Returns `LStatusOK` if successful. If an error occurs, `LStatus` contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LBadFile</code></td>
<td><code>TDBFile</code> is NULL.</td>
</tr>
</tbody>
</table>

**Parameters**

- `TDBFile` The specified TDB file.

**Example**

```c
/* Get the active TDB file */
LFile TDBFile = LFile_GetVisible();
if(Assigned(TDBFile))
{
    /* Set the User Data */
    LCell MyCell = LCell_GetVisible();
    LFile_SetUserData(TDBFile, MyCell);

    /* Do some processing on the TDB file */
    ...

    /* Clear the User Data */
    LFile_ClearUserData(TDBFile);
}
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

LFile_DisplayCellBrowser

```
void LFile_DisplayCellBrowser( LFile file );
```

**Description**

Displays the cell browser for the specified file.

**Parameters**

- `file` 
  Specified file.

**See Also**

“LFile” (page 1736), “Interface Functions” (page 1075)
LFile_SetLastCurrent

LStatus LFile_SetLastCurrent( LFile file, LCell cell );

Description

Sets the last open cell in the specified file.

Return Values

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

Parameters

file Specified file.

cell Specified cell.

See Also

**LFile_GetDesignRuleFlags**

```c
LStatus LFile_GetDesignRuleFlags( LFile file, LDesignRuleFlags *pDRCFlags );
```

**Description**

Gets DRC flags.

**Return Values**

Returns LStatusOK if successful or LBadParameter if an error occurred.

**Parameters**

- `file` Current file.
- `pDRCFlags` Pointer to LDesignRuleFlags.

**Example**

```c
LAmbiguousFillType GetActionOnPolygonsWithAmbiguousFills( LFile file )
{
    LDesignRuleFlags drcFlags; // LDesignRuleFlags structure
    LFile_GetDesignRuleFlags( file, &drcFlags ); // get current flags
    return drcFlags.PolygonsWithAmbiguousFills; // return one of the values
}
```

**Note:**

Note that this function is superseded in L-Edit V10 and later.

**See Also**

“LDesignRuleFlags” (page 1721), “LAmbiguousFillType” (page 1695),
“LFile_SetDesignRuleFlags” (page 1243)
LFile_SetDesignRuleFlags

LStatus LFile_SetDesignRuleFlags( LFile file, LDesignRuleFlags *pDrcFlags );

Description

Sets DRC flags.

Return Values

Returns LStatusOK if successful or LBadParameter if an error occurred.

Parameters

**file**

Specified file.

**pDRCFlags**

Pointer to LDesignRuleFlags.

Example

```c
void SetFlagIgnoredObject( LFile file, LBoolean flagIgnored )
{
    LDesignRuleFlags drcFlags; // LDesignRuleFlags structure
    LFile_GetDesignRuleFlags( file, &drcFlags ); // get current flags
    drcFlags.FlagIgnoredObjects = flagIgnored; // change one of the flags
    LFile_SetDesignRuleFlags(file, &drcFlags); // modify current flags
}
```

**Note:** Note that this function is superseded in L-Edit V10 and later.

See Also

“LDesignRuleFlags” (page 1721), “LAmbiguousFillType” (page 1695),
“LFile_GetDesignRuleFlags” (page 1242)
LFile_GetResolvedFileName

LStatus LFile_GetResolvedFileName( LFile pTDBFile, const char* szRelativeFileName, char* szAbsoluteFileNameBuffer, int iBufferSize );

Description

Resolves a path that is relative to a TDB file to an absolute path. Absolute paths are not modified. Use this function to resolve a path that might be relative to the TDB file, such as the extract definition file, DRC error file, or a file returned from LDialog_File.

Return Values

LBufferDataTooSmall if the buffer was too small, LStatusOK otherwise.

Parameters

pTDBFile The TDB file that the path might be relative to.

szRelativeFileName Path that is either absolute or relative to the TDB file.

szAbsoluteFileNameBuffer Buffer in which to store the absolute path.

iBufferSize Size of the buffer.

Example

LFile pTDBFile = LFile_GetVisible();
if(Assigned(pTDBFile))
{
    // Get the filename.
    char szFileName[512] = "\0";
    LDialog_File(NULL, "Import Filename", szFileName, "Text file (*.txt)|*.lys|All Files (*.*)|*.*||", 0, "Import filename:", NULL, "*.txt", NULL, pTDBFile);

    // Check if the user cancelled the dialog.
    if(strlen(szFileName) > 0)
    {
        char szAbsoluteFileName[512] = "\0";
        if(LFile_GetResolvedFileName(pTDBFile, szFileName, szAbsoluteFileName, 511) == LStatusOK)
        {
            // More Processing
            // ...
        }
        else
        {
            // ERROR: Not enough space in the buffer.
        }
    } // endif(strlen(szFileName) > 0)
} // endif(Assigned(pTDBFile))
Version

Available in L-Edit 8.2 and later versions.

See Also

“File Functions” (page 1193), “LDialogItem” (page 1723), “LFile” (page 1736)
**LFile_GetVisible**

```c
LFile LFile_GetVisible( void );
```

**Description**

Gets the visible (active) TDB file.

**Return Values**

Returns a pointer to the active TDB file; otherwise NULL. If the visible file is a text file, then NULL is returned.

**Example**

```c
LFile pFile = LFile_GetVisible();
if( Assigned( pFile ) )
{
    char szFileName[256];
    LFile_GetName( pFile, szFileName, 255 );
    // More Processing
    // ...
}
```

**Version**

Available in L-Edit 8.2 and later versions.

**See Also**

“File Functions” (page 1193).
LFile_IntUtoLocU

double LFile_IntUtoLocU( LFile pFile, LCoord lcValue )

Description

Returns the value (lcValue) in Locator Units.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFile</td>
<td>Specified TDB file.</td>
</tr>
<tr>
<td>lcValue</td>
<td>Value in Internal Units.</td>
</tr>
</tbody>
</table>

Example

LFile pFile = LFile_GetVisible();
if(Assigned(pFile))
{
   double dWidth = LFile_IntUtoLocU(pFile, 1000);
   // More Processing
   // ...
}

Version

Available in L-Edit 8.2 and later versions.

Note: Note that this function is obsolete in L-Edit V10 and later.

See Also

LFile_LocUtoIntU

LCoord LFile_LocUtoIntU( LFile pFile, double dValue )

Description

Converts a value in Locator Units to Internal Units based on the grid mapping of a TDB file. This function will round the value if it cannot represent the value exactly in Internal Units. For example, if the mapping is 1 LU = 10 IU, and the value is 0.25 then it will be rounded to 0.3.

Return Values

Returns the value (dValue) in Internal Units.

Parameters

- **pFile**: Specified TDB file.
- **dValue**: Value in Locator Units.

Example

```c
LFile pFile = LFile_GetVisible();
if( Assigned( pFile ) )
{
    LCoord lcWidth = LFile_LocUtoIntU( pFile, 25.75 );
    // More Processing
    // ...
}
```

Version

Available in L-Edit 8.2 and later versions.

Note: Note that this function is obsolete in L-Edit V10 and later.

See Also

LFile_GetDisplayUnitInfo

```c
LStatus LFile_GetDisplayUnitInfo( LFile file, LDisplayUnitInfo *pDispUnitInfo );
```

Description

Get a structure containing information about the display units.
LFile_SetDisplayUnit

LStatus LFile_SetDisplayUnit( LFile file, const char* szDispUnitName );

Description

Configure the display unit settings.
**LFile_IntUtoDispU**

```c
double LFile_IntUtoDispU( LFile pFile, LCoord lcValue );
```

**Description**

Convert a dimension in Internal Units to the equivalent number of Display Units.
LFile_DispUtoIntU

LCoord LFile_DispUtoIntU( LFile pFile, double dValue );

Description

Convert a dimension in Display Units to the equivalent number of Internal Units.
LFile_IntUtoMicrons

double LFile_IntUtoMicrons( LFile pFile, LCoord lcValue );

Description

Convert a dimension in Internal Units to the equivalent number of microns.
LFile_MicronsToIntU

```
LCoord LFile_MicronsToIntU( LFile pFile, double dValue );
```

Description

Convert a dimension in microns to the equivalent number of Internal Units.
Chapter 34: UPI Functions Reference

LFile_GetCurrentLayerPalette

const char* LFile_GetCurrentLayerPalette( LFile pTDBFile );

Description

Returns the name of the currently active layer palette.

Use this and the next three functions to run and add newly created layers to custom Layer Palette filters.
LFile_SetCurrentLayerPalette

LStatus LFile_SetCurrentLayerPalette( LFile pTDBFile, const char *cszPalette );

Description

Activates the named layer palette, which must already exist.
LFile_GetNextLayerPalette

const char* LFile_GetNextLayerPalette( LFile pTDBFile, const char* cszPalette );

Description

Returns the name of the next layer palette. Pass NULL for cszPalette to get the first palette name; returns NULL after the last one.
LFile_DefineLayerPalette

LStatus LEDITAPI LFile_DefineLayerPalette( LFile pTDBFile, const char *cszPalette );

Description

Creates or overwrites the palette with the given name, using the current layer settings.
**LFile_GetTopLevelLibraryName**

```c
LStatus LFile_GetTopLevelLibraryName( LFile pTDBFile, char* szName, const int cnMaxname );
```

**Description**

Gets the top library name.

**Return Values**

Returns L.StatusOK if successful or an error code.

**Parameters**

- **pTDBFile**
  Specified TDB file.
- **szName**
  String (buffer) containing name of the top library.
- **cnMaxname**
  Maximum length allowed for the szName.

**Version**

Available in L-Edit 16.00 and later versions.
Cell Functions

Cell functions allow the user to manipulate an individual cell in an L-Edit design file. Subcategories of cell functions include:

- "Instance Functions" (page 1324)
- "Entity Functions" (page 1345)
- "Object Functions" (page 1371)

General cell functions are listed below:

- LCell_New” (page 1261)
- LCell_Copy” (page 1263)
- LCell_GetFile” (page 1266)
- LCell_GetList” (page 1267)
- LCell_GetLock” (page 1269)
- LCell_GetName” (page 1271)
- LCell_GetAuthor” (page 1273)
- LCell_GetOrganization” (page 1275)
- LCell_GetInfoText” (page 1277)
- LCell_GetView” (page 1286)
- LCell_GetUserData” (page 1292)
- LCell_GetShowInLists” (page 1294)
- LCell_ClearUndoLists” (page 1308)
- LCell_IsChanged” (page 1285)
- LCell_GetMbb” (page 1288)
- LCell_CalcMBB” (page 1314)
- LCell_RemoveMarker” (page 1316)
- LCell_GetParameter” (page 1309)
- LCell_RunDRCEx01” (page 1305)
- LCell_BooleanOperation” (page 1318)
- LCell_RunDRCEx01” (page 1305)
- LCell_Slice” (page 1321)
- LCell_GenerateLayersEx830” (page 1295)
- LCell_RunDRCEx00” (page 1303)
- LCell_Delete” (page 1262)
- LCell_Find” (page 1264)
- LCell_GetNext” (page 1268)
- LCell_SetLock” (page 1270)
- LCell_SetName” (page 1272)
- LCell_SetAuthor” (page 1274)
- LCell_SetOrganization” (page 1276)
- LCell_SetInfoText” (page 1278)
- LCell_SetView” (page 1287)
- LCell_SetUserData” (page 1293)
- LCell_ClearUserData” (page 1291)
- LCell_SetShowInLists” (page 1312)
- LCell_Flatten” (page 1290)
- LCell_SetChanged” (page 1301)
- LCell_GetMbbAll” (page 1289)
- LCell_AddMarker” (page 1315)
- LCell_RemoveAllMarkers” (page 1317)
- LCell_GetVersion” (page 1281)
- LCell_SetVersion” (page 1282)
- LCell_GetCreatedTime” (page 1283)
- LCell_GetModifiedTime” (page 1284)
- Obsolete
- LCell_GenerateLayersEx99” (page 1297)
- LCell_GenerateLayers_v10_00” (page 1298)
**LCell_New**

```c
LCell LCell_New( LFile file, char *name );
```

**Description**

Creates a new cell in the specified file.

**Return Values**

Returns a pointer to the newly created cell if successful; otherwise returns NULL.

**Parameters**

- `file`  
  File where new cell need to be created.
- `name`  
  Name of the new cell.

**Example**

```c
LFile pFile = LFile_GetVisible();
if( Assigned( pFile ) )
{
    LCell pCell = LCell_New( pFile, "MyCell" );
    if ( pCell )
    {
        // More Processing
        // ...
    }
}
```

**See Also**

**LCell_Delete**

```c
LStatus LCell_Delete( LCell cell );
```

**Description**

Deletes the specified cell from the current file.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

**Parameter**

`cell` Cell to be deleted.

**See Also**

**LCell_Copy**

```c
LStatus LCell_Copy( LFile sourceFile, LCell sourceCell, LFile destFile, char *destCellName );
```

**Description**

Copies a cell from one file (the "source" file) to another (the "destination" file—possibly the same) with a new name. If a cell with the new name already exists in the destination file, it is overwritten.

**Return Values**

Returns **LStatusOK** if no name collision occurs, **LCellOverWritten** if there is a collision. Returns **LBadParameters** if null parameters are passed or if **sourceCell** does not belong to **sourceFile**. Returns **LLayerMapsDifferent** if the layer maps in **sourceFile** and **destFile** are not the same.

**Parameters**

- **sourceFile**
  Source file.
- **sourceCell**
  Cell to be copied.
- **destFile**
  Destination file.
- **destCellName**
  Name of the new cell.

**See Also**

“**LStatus**” (page 1792), “**LObject**” (page 1769), “**LFile**” (page 1736), “**Cell Functions**” (page 1260)
LCell_Find

LCell LCell_Find( LFile file, const char* name );

Description

Searches for a cell of the specified name in the specified file.

Return Values

Returns a pointer to the cell if found; otherwise returns NULL.

Parameters

file File to search.
name Cell name to search for.

Example

LFile pFile = LFile_GetVisible();
if( Assigned( pFile ) )
{
    LCell pCell = LCell_Find( pFile, "MyCell" );
    if ( !pCell )
        LDialog_AlertBox( "Cell not found" );
}

See Also

LCell_FindEx

LCell LCell_FindEx( LFile file, const char* name );

Description

Use LCell_FindEx in T-Cell code and UPI code to find a cell. This function will look in XrefFiles for the cell if it doesn’t find it in the current file and will also copy the cell over to the file.

Return Values

Returns a pointer to the cell if found; otherwise returns NULL.

Parameters

file File to search.

name Cell name to search for.
LCell_GetFile

```c
LFile LCell_GetFile( LCell cell );
```

**Description**

Returns a pointer to the parent file of the specified cell. Note: this function searches the entire list of cells in each open file, in order to match ‘cell’. This is inefficient in the case of designs with many cells. It is usually a better idea to explicitly keep track of the file from which ‘cell’ was obtained.

**Return Values**

Returns a pointer to the file if found; otherwise returns NULL.

**Parameters**

- `cell` Specified cell.

**Example**

```c
LCell pCell = LCell_GetVisible();
if ( pCell )
{
    LFile pFile = LCell_GetFile( pCell );
    if ( pFile != LFile_GetVisible() )
        LDialog_AlertBox( "this should never happen!" );
}
```

**See Also**

LCell_GetList

LCell LCell_GetList( LFile file );

Description

Gets a list of cells in the specified file.

Return Values

Returns a pointer to the head of the cell list if successful; otherwise returns NULL.

Parameters

file Specified file.

Example

LFile pFile = LFile_GetVisible();
int nCount = 0;
LCell pCell;
for ( pCell = LCell_GetList(pFile); pCell; pCell = LCell_GetNext(pCell) )
  nCount ++;
LDialog_MsgBox( LFormat( "%d cells", nCount ) );

See Also

LCell_GetNext

LCell LCell_GetNext( LCell cell );

Description

Gets the next cell in the current file’s list of cells after the specified cell.

Return Values

Returns a pointer to the next cell if successful; otherwise returns NULL.

Parameters

*cell*  
Specified cell.

Example

```
LFile pFile = LFile_GetVisible();
int nCount = 0;
LCell pCell;
for ( pCell = LCell_GetList(pFile); pCell; pCell = LCell_GetNext(pCell) )
    nCount ++;
LDialog_MsgBox( LFormat( "%d cells", nCount ) );
```

See Also

LCell_GetLock

```c
int LCell_GetLock( LCell cell );
```

**Description**

Finds out if a cell is locked or not. If not locked, and database is OpenAccess, tries to obtain a reservation on the cell.

**Return Values**

Returns zero if the specified cell is unlocked, and if database is OpenAccess a reservation was successfully obtained; otherwise returns a nonzero value.

**Parameters**

- `cell` Cell to be checked.

**See Also**

“LObject” (page 1769), “Cell Functions” (page 1260)
LCell_SetLock

```c
int LCell_SetLock( LCell cell, int set );
```

**Description**

Locks or unlocks the specified cell.

**Return Values**

Returns zero if the specified cell has been unlocked; otherwise returns a nonzero value.

**Parameters**

- `cell` Cell to be locked or unlocked.
- `set` Value that determines the cell’s new status: zero unlocks; anything else locks.

**See Also**

“LObject” (page 1769), “Cell Functions” (page 1260)
**LCellGetName**

```c
char* LCellGetName( LCell cell, char* name, const int maxlen );
```

**Description**

Gets the name of the specified cell.

**Return Values**

Returns a pointer to the string `name` if successful; otherwise returns NULL.

**Parameters**

- `cell` Cell whose name is to be retrieved.
- `name` String containing the name text.
- `maxlen` Maximum length allowed for `name`.

**Example**

```c
LCell pCell = LCell_GetVisible();
if ( pCell )
{
    char name[MAX_CELL_NAME];
    if ( LCellGetName( pCell, name, MAX_CELL_NAME ) )
        LDialog_MsgBox( name ); // print it out
}
```

**See Also**

- "LObject" (page 1769), "LFileGetName" (page 1208), "LInstanceGetName" (page 1334), "Cell Functions" (page 1260)
LCell_SetName

LStatus LCell_SetName( LFile file, LCell cell, const char* name );

Description

Sets the name of the specified cell in the specified file.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file File whose cell is being renamed.

cell Cell to be (re)named.

name New cell.

See Also

LCell_GetAuthor

char* LCell_GetAuthor( LCell cell, char* author, const int maxlen );

Description

Gets the text of the string author for the specified cell.

Return Values

Returns a pointer to the string author if successful; otherwise returns NULL.

Parameters

- cell: Cell whose author is to be retrieved.
- author: String containing the author text.
- maxlen: Maximum length allowed for author.

See Also

**LCell_SetAuthor**

```c
char* LCell_SetAuthor(LCell cell, char* author);
```

**Description**

Sets the text of the string `author` for the specified cell.

**Return Values**

Returns a pointer to the structure containing the string `author` if successful; otherwise returns NULL.

**Parameters**

- `cell` Cell whose author is to be set.
- `author` String containing the `author` text.

**See Also**

“LObject” (page 1769), “Cell Functions” (page 1260)
LCell_GetOrganization

    char* LCell_GetOrganization( LCell cell, char* org, const int maxlen );

Description

Gets the organization text associated with the specified cell.

Return Values

Returns a pointer to the cell organization buffer if successful; otherwise returns NULL.

Parameters

    cell  Cell whose organization is to be retrieved.
    org   String containing the organization text.
    maxlen Maximum length allowed for org.

See Also

“LObject” (page 1769), “Cell Functions” (page 1260)
LCell_SetOrganization

char* LCell_SetOrganization( LCell cell, char* org );

Description

Sets the text of the organization field in the information summary of the specified cell. A NULL value may be given.

Return Values

Returns a pointer to the string containing the organization text if successful; otherwise returns NULL.

Parameters

cell Cell whose organization is to be set.
org String containing the organization text.

See Also

“LObject” (page 1769), “Cell Functions” (page 1260)
LCell_GetInfoText

```
char* LCell_GetInfoText(LCell cell, char* info, const int maxlen);
```

**Description**

Gets the text of the information field in the information summary of the specified cell.

**Return Values**

Returns a pointer to the cell info buffer if successful; otherwise returns NULL.

**Parameters**

- **cell**: Cell whose information is to be retrieved.
- **info**: String containing the information text.
- **maxlen**: Maximum length allowed for info.

**See Also**

“LObject” (page 1769), “Cell Functions” (page 1260)
**LCell_SetInfoText**

```c
char* LCell_SetInfoText( LCell cell, char* info );
```

**Description**

Sets the text of the information field in the information summary of the specified cell. A NULL value may be given.

**Return Values**

Returns a pointer to the string `info` if successful; otherwise returns NULL.

**Parameters**

- `cell` Cell whose information is to be set.
- `info` String containing the new information text.

**See Also**

“LObject” (page 1769), “Cell Functions” (page 1260)
**LCell_MakeLogo**

```c
LStatus LCell_MakeLogo( LCell pCell, const char* szLogoString, LCoord nTextSize, LLayer pLayer, LBoolean bPrintDate, LBoolean bPrintMaskCopyright, LCoord CenterPointX, LCoord CenterPointY, LBoolean bPrintCompanyLogo, LBoolean bSnapToGrid, LBoolean bCreateAsInstance, const char* cszLogoLibrary, const char* cszLogoCell, const char* cszLogoView, const char* cszCharacterSetFileName );
```

**Description**

In a specified cell, generates text or a logo with given the parameters.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- **pCell**  
  Cell containing the port.

- **szLogoString**  
  String to generate as characters. The following special characters are allowed:
  - &c - Center Justify
  - &l - Left justify
  - &r - Right justify
  - &d - Current date
  - &t - Company logo
  - &m - Mask copyright
  - \n - newline

- **nTextSize**  
  Scales the layout text size. The MBB of the alphabet cells is scaled to the value specified (in display units).

- **pLayer**  
  Layer to generate the layout text on. This layer must exist.

- **bPrintDate**  
  Prints the current date.

- **bPrintMaskCopyright**  
  Prints the mask copyright.

- **bUnderline**  
  Underlines the layout text string and date.

- **CenterPointX, CenterPointY**  
  Center point of resulting polygons.

- **bPrintCompanyLogo**  
  Prints the company logo.

- **bSnapToGrid**  
  Snaps the resulting polygons to the manufacturing grid.

- **bCreateAsInstance**  
  Generates the layout text in a new cell, which will be placed into the current cell as an instance.
See Also

LCell_GetVersion

```c
LStatus LCell_GetVersion( LCell cell, long *major, long *minor );
```

Description

Copies the version numbers of the specified cell to the variables pointed to by the major and minor values.

Return Values

Returns a pointer to a long integer containing the version number (in *major and *minor) and LStatusOK if successful; otherwise returns NULL.

Parameters

- `cell` : Cell whose information is to be retrieved.
- `major` : String containing the major version number.
- `minor` : String containing the minor version number.

See Also

**LCell_SetVersion**

```c
LStatus LCell_SetVersion(LCell cell, long major, long minor);
```

**Description**

Sets the field values of the specified cell to the variables pointed to in a long integer containing the version numbers.

**Return Values**

Returns LStatusOK if successful; otherwise returns NULL.

**Parameters**

- `cell` Cell whose information is to be set.
- `major` String containing the major version number.
- `minor` String containing the minor version number.

**See Also**

**LCell_GetCreatedTime**

```c
long LCell_GetCreatedTime ( LCell pCell );
```

**Description**

Gets the creation date and time associated with the specified cell.

**Return Values**

Returns the creation time of the specified cell.

The time value is expressed in seconds from Jan 1, 1970, and is compatible with the `time_t` value returned by `time()` (in compiled UPI scripts only).

**Parameters**

- `pCell` Cell to be queried.

**See Also**

- “LObject” (page 1769), “Cell Functions” (page 1260)
LCell_GetModifiedTime

```c
long LCell_GetModifiedTime ( LCell pCell );
```

**Description**

Gets the most recent modification date and time for the specified cell.

**Return Values**

Returns the modified date and time of the specified cell.

The time value is expressed in seconds from Jan 1, 1970, and is compatible with the `time_t` value returned by `time()` (in compiled UPI scripts only).

**Parameters**

- `pCell`  
  Cell to be queried.

**See Also**

- "LObject" (page 1769), "Cell Functions" (page 1260)
LCell_IsChanged

    int LCell_IsChanged( LCell pCell )

Description

Checks the specified cell to determine if it has been changed since it was last saved.

Return Values

The function returns 1 if the cell has been changed or 0 if it has not.

Parameters

  pCell          Cell to be checked.

See Also

“LObject” (page 1769), “Cell Functions” (page 1260)
LCell_GetView

LRect LCell_GetView( LCell cell );

Description

Gets the coordinates of the rectangle that defines the current view of the specified cell.

Return Values

Returns the coordinates of the viewing rectangle if successful; otherwise returns a rectangle whose coordinates are all zeros.

Parameters

cell

Cell whose viewing rectangle is needed.

See Also

**LCell_SetView**

```
LStatus LCell_SetView( LCell cell, LRect view );
```

**Description**

Sets the coordinates of the rectangle that defines the current view of the specified cell.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

**Parameters**

- `cell`  
  Cell whose viewing rectangle needs to be set.
- `view`  
  New viewing rectangle.

**See Also**

“**LStatus**” (page 1792), “**LTransform**” (page 1801), “**LObject**” (page 1769), “**Cell Functions**” (page 1260)
LCell_GetMbb

```c
LRect LCell_GetMbb( LCell cell );
```

**Description**

Gets the coordinates of the rectangle representing the minimum bounding box (Mbb) of the specified cell, not including port text.

**Return Values**

Returns the Mbb if successful; otherwise returns a rectangle whose coordinates are all zeros.

**Parameters**

- `cell` Cell whose Mbb is to be found.

**See Also**

LCell_GetMbbAll

LRect LCell_GetMbbAll( LCell cell );

Description

Gets the coordinates of the rectangle representing the minimum bounding box (Mbb) of the specified cell, including port text.

Return Values

Returns the MbbAll rectangle if successful; otherwise returns a rectangle whose coordinates are all zeros.

Parameters

cell Cell whose MbbAll is to be found.

See Also

LCell_Flatten

LCell LCell_Flatten( LCell cell );

Description

Flattens the specified cell.

Return Values

Returns a pointer to the flattened cell if successful; otherwise returns NULL.

Parameters

cell

Cell to be flattened.

See Also

“LObject” (page 1769), “Cell Functions” (page 1260)
LCell_ClearUserData

LStatus LCell_ClearUserData( LCell cell );

Description

Sets the user-defined data pointer on the specified cell to NULL without freeing it.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadCell</td>
<td>cell is NULL.</td>
</tr>
</tbody>
</table>

Parameters

- **cell** The specified cell.

Examples

```c
/* Get the active cell */
LCell MyCell = LCell_GetVisible();
if(assigned(MyCell))
{
    /* Set the user data */
    LFile TDBFile = LFile_GetVisible();
    LCell_SetUserData(MyCell, TDBFile);

    /* Do some processing on MyCell */
    ...

    /* Clear the user data */
    LCell_ClearUserData(MyCell);
}
```

Version

Available in L-Edit 8.4 and later versions.

See Also

LCell_GetUserData

```
void* LCell_GetUserData( LCell cell );
```

**Description**

Gets a pointer to user-defined data associated with the specified cell.

**Return Values**

Returns a pointer to the user data if successful; otherwise returns NULL.

**Parameter**

- `cell` Cell whose user-defined data is needed.

**Example**

```c
/*Declare user-defined data to be stored in a cell*/
typedef struct {
    int x;
    double y;
    float z;
} CellUserDataRec;

CellUserDataRec cd, *pd=NULL;

/*The Cell Pointer*/
LCell cell;

/*Get a pointer to the currently open cell*/
cell = LCell_GetVisible();

/*Fill in data into CellUserDataRec*/
cd.x = 1; cd.y = 2.0; cd.z = 1.5;

/*Store cd into cell’s data pointer*/
LCell_SetUserData( cell, (void *) (&cd));

/*Get the data back from cell’s data pointer into pd*/
pd = (CellUserDataRec *) LCell_GetUserData( cell );

/*pd now points to the user-defined data*/
```

**See Also**

“LObject” (page 1769), “Cell Functions” (page 1260)
LCell_SetUserData

LStatus LCell_SetUserData( LCell cell, void* dataPointer );

Description

Uses a data pointer within a cell to associate user-defined data with the cell. Data can be integer, string, or any other type.

Return Values

Returns L.Status_OK if successful. If an error occurs, L.Status contains the error value.

Parameter

cell

Cell which will contain the user-defined data.

Example

/*Declare user-defined data to be stored in a cell*/
typedef struct {
    int x;
    double y;
    float z;
} CellUserDataRec;

CellUserDataRec cd;

/*The Cell Pointer*/
LCell cell;

/*Get a pointer to the currently open cell*/
cell = LCell_GetVisible();

/*Fill in data into CellUserDataRec*/
cd.x = 1; cd.y = 2.0; cd.z = 1.5;

/*Store cd into cell’s data pointer*/
LCell_SetUserData( cell, (void *) (&cd));

See Also

LCell_DeleteUserData

LStatus LCell_DeleteUserData(LCell cell);

Description

Deletes the user-defined expansion pointer in the specified cell.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell Specified cell.

See Also

LCell_GenerateLayersEx830

```c
LStatus LCell_GenerateLayersEx00( LCell pCell, int iBinSize,
   LLayer pLayer, LBoolean bDeletePreviousDerivedLayers,
   LBoolean bMergeObjectsAfterGeneration )
```

**Description**

Generates a layer or all layers in the specified cell.

**Return Values**

- **LStatusOK** if successful. If an error occurs, LStatus contains the error value with possible values:
  - LBadCell — pCell is NULL
  - LBadParameters — iBinSize is <= 0

**Parameters**

- **pCell** Cell to generate the layers in.
- **iBinSize** Bin size for generating layers.
- **pLayer** Layer to generate. If pLayer is NULL then all layers are generated.
- **bDeletePreviousDerivedLayers** If LTRUE, all existing derived layers will be deleted.
- **bMergeObjectsAfterGeneration** If LTRUE, causes objects on the same generated layer to be merged upon completion of the process, which can significantly increase the processing time for more complex layouts.

**Example**

```c
LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
   LFile pFile = LCell_GetFile(pCell);
   if(Assigned(pFile))
   {
      LLayer pLayer = LLayer_Find(pFile, "ndiff");
      if(Assigned(pLayer))
      {
         if(LCell_GenerateLayersEx00(pCell,
            LFile_LocUtoIntU(pFile, 100), pLayer, LTRUE, LFALSE) == LStatusOK)
         {
             // More Processing
             // ... } // endif(Assigned(pLayer))
      } else
      {
         LDialog_AlertBox("Cannot find layer "ndiff"\n");
      } // endif(Assigned(pLayer))
} // endif(Assigned(pCell))
```
Note: Note that this function is obsolete in L-Edit V10 and later.

See Also

“LCell_GenerateLayers_v11_10” (page 1300), “LCell_GenerateLayers_v10_00” (page 1298),
“LCell_GenerateLayersEx99” (page 1297), “LCell” (page 1700), “LLayer” (page 1755),
LCell_GenerateLayersEx99

LStatus LCell_GenerateLayersEx99( LCell pCell, int iBinSize, LLayer pLayer );

Description

Generates the layer or layers in the specified cell. To generate all layers, set pLayer to NULL.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

  pCell        Cell to generate the layers in.
  iBinSize     Bin size for generating layers.
  pLayer       Layer to generate. If pLayer is NULL then all layers are generated.

Note:

Note that this function is obsolete in L-Edit V10 and later.

See Also

LCell_GenerateLayers_v10_00

LStatus LCell_GenerateLayers_v10_00( LCell pCell, LLayer* ArrayOfLayers, unsigned int nNumberOfLayers, LBoolean bClearAllGeneratedLayers, LBoolean bMergeObjectsAfterGeneration );

Description

Generates a layer or all layers in the specified cell.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value with possible values:

LBadCell — pCell is NULL

LBadParameters — iBinSize is <= 0

Parameters

- **pCell**
  - Cell to generate the layers in.
- **ArrayOfLayers**
  - Layer(s) to generate.
- **nNumberOfLayers**
  - Number of layers to generate.
- **bClearAllGeneratedLayers**
  - If LTRUE, all existing derived layers will be deleted.
- **bMergeObjectsAfterGeneration**
  - If LTRUE, causes objects on the same generated layer to be merged upon completion of the process, which can significantly increase the processing time for more complex layouts.

Example

LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    LFile pFile = LCell_GetFile(pCell);
    if(Assigned(pFile))
    {
        LLayer pLayer = LLayer_Find(pFile, "ndiff");
        if(Assigned(pLayer))
        {
            if(LCell_GenerateLayers_v10_00( pCell, &pLayer, 1, LTRUE, LFALSE) == LStatusOK)
            {
                // More Processing
                // ...
            }
            else
                LDialog_AlertBox("Cannot find layer "ndiff"";
        } // endif(Assigned(pFile))
    } // endif(Assigned(pFile))
}
See Also

“LCell_GenerateLayers_v11_10” (page 1300), “LCell_GenerateLayersEx99” (page 1297),
“LCell_GenerateLayersEx830” (page 1295), “LCell” (page 1700), “LLayer” (page 1755),
LCell_GenerateLayers_v11_10

LStatus LCell_GenerateLayers_v11_10(
    LCell pCell,
    const char* szCommandFile,
    const char** pszArrayOfLayerNames,
    unsigned int nNumberOfLayers
    LBoolean bMergeObjectsAfterGeneration);

Description

This function works similarly to the Generate Layers dialog but instead of showing the dialog, the layers to be generated are input through the pszArrayOfLayerNames parameter. Layers are generated in the specified cell according to the following rules.

If a layer name does not exist in the command file, it is ignored. If a layer name does not exist in L-Edit, it is created with its type as external.

If the layer exists in L-Edit and its type is external, it clears the layer before bringing in the generated layers. If the layer exists in L-Edit and its type is not external, the layer is still generated but the name is modified with _1” appended (using appropriate name collision checking and incrementing).

Return Values

- LStatusOK if successful. If an error occurs, LStatus contains the error value with possible values:
- LBadCell—if pCell is NULL
- LBadParameters—if iBinSize is <= 0
- LCopyProtViolation—if no HiPer license exists
- LOpenError—if szCommandFile does not exist
- LUserAbort—if the user aborted the generate layers operation

Parameters

- pCell Cell in which to generate the layers.
- CommandFile Command file to check.
- ArrayOfLayerNames Layer name(s) to generate.
- nNumberOfLayers Number of layers to generate.
- bMergeObjectsAfterGeneration If LTRUE, causes objects on the same generated layer to be merged upon completion of the process, which can significantly increase the processing time for more complex layouts.

See Also

LCell_SetChanged

void LCell_SetChanged( LCell pCell )

Description

Marks the cell as changed. This will also increment the minor version.

Parameters

pCell The cell to mark as changed.

Example

if(Assigned(pCell))
{
    // Do some processing on pCell.
    // ...

    // Mark the file as changed.
    LCell_SetChanged(pCell);
}

Version

Available in L-Edit 8.2 and later versions.

See Also

**LCell_RunDRC**

```c
LStatus LCell_RunDRC( LCell pCell, const LRect* pDRCArea,
                        unsigned int *pnNumErrors );
```
LCell_RunDRCEx00

```c
LStatus LCell_RunDRCEx00( LCell pCell, LRect *pDRCArea, LCoord lcBinSize, const char* szErrorFile, LBoolean bWriteErrorPorts, LBoolean bWriteErrorObjects )
```

Description

Runs DRC on the entire cell or a specified area of a cell. You can specify the bin size, the DRC error file, and whether to place error ports and objects during DRC. If Quietmode is on, then the RUN DRC dialog and warning dialogs will not appear.

**Note:** Note that this function is superseded by “LCell_RunDRCEx01” (page 1305).

Return Values

- **LStatusOK** if successful. If an error occurs, **LStatus** contains the error value with possible values:
  - **LBadCell** — pCell is NULL
  - **LSystemError** — L-Edit could not find an active cell layout window or an open one.
  - **LUserAbort** — The user canceled DRC while it was running.

Parameters

- **pCell** Cell to run DRC on.
- **pDRCArea** Rectangular area to perform DRC on. If **pDRCArea** is NULL, then DRC is performed on the entire cell.
- **lcBinSize** Bin size in Internal Units.
- **szErrorFile** Name of the file to write DRC error to. If **szErrorFile** is NULL, no errors are written to a file.
- **bWriteErrorPorts** Instructs L-Edit to place an error port on the specified error layer at the location of each DRC violation. An error port consists of the name of the violated design rule and a bracketed expression; the expression indicates the spacing or nature of the error and the rule distance.
- **bWriteErrorObjects** Instructs L-Edit to place marker objects on the specified error layer at the location of each violation.

Example

```c
LCell pCell = LCell_GetVisible();
if(Assigned(pCell)) {
    LFile pFile = LCell_GetFile(pCell);
    ... // Call LCell_RunDRCEx00 with parameters
}```
if(Assigned(pFile))
{
    if(LCell_RunDRCEx00(pCell, NULL, LFile_LocUtoIntU(pFile, 100),
        NULL, LTRUE, LFALSE) == LStatusOK)
        {
            // More Processing
            // ...
        }
} // endif(Assigned(pFile))

} // endif(Assigned(pCell))

Version

Available in L-Edit 8.2 and later versions.

Note: Note that this function is obsolete in L-Edit V10 and later.

See Also

“DRC Functions” (page 1622), “LCell” (page 1700), “LRect” (page 1783), “LCoord” (page 1707),
**LCell_RunDRCEx01**

```c
LStatus LCell_RunDRCEx01( LCell pCell, LRect *pDRCArea, LCoord lcBinSize, const char* szErrorFile, int flags)
[also LBoolean bWriteErrorPorts, LBoolean bWriteErrorObjects,
 LBoolean bWriteErrors, LBoolean bWriteTimingStatistics);
```

**Description**

LCell_RunDRCEx01 is similar to LCell_RunDRCEx00 except that it supports timing statistics and all flags should be passed through the flags parameter.

Runs DRC on the entire cell or a specified area of a cell. You can specify the bin size, the DRC error file, and whether to place error ports and objects during DRC. If Quietmode is on, then the RUN DRC dialog and warning dialogs will not appear.

**Note:**
This function supersedes “LDRC_Run” (page 1634) and “LCell_RunDRCEx00” (page 1303). LCell_RunDRCEx830 can be used interchangeably with LCell_RunDRCEx01.

**Return Values**

- **LStatusOK** if successful. If an error occurs, **LStatus** contains the error value with possible values:
  - **LBadCell** — pCell is NULL
  - **LSystemError** — L-Edit could not find an active cell layout window or an open one.
  - **LUserAbort** — The user canceled DRC while it was running.
Parameters

- **pCell**: Cell to run DRC on.
- **pDRCArea**: Pointer to an `LRect` structure that specifies a rectangular area to perform DRC on. If `pDRCArea` is NULL, then DRC is performed on the entire cell.
- **lcBinSize**: Bin size in Internal Units.
- **szErrorFile**: Name of the file to write DRC error to. If `szErrorFile` is NULL, no errors are written to a file.
- **bWriteErrors**: Instructs L-Edit to write errors to the specified text file.
- **bWriteErrorPorts**: Instructs L-Edit to place an error port on the specified error layer at the location of each DRC violation. An error port consists of the name of the violated design rule and a bracketed expression; the expression indicates the spacing or nature of the error and the rule distance.
- **bWriteErrorObjects**: Instructs L-Edit to place marker objects on the specified error layer at the location of each violation.
- **bWriteTimingStatistics**: Instructs L-Edit to write the elapsed time for each layer derivation and each DRC rule check to a text file.

Example

```c
LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    LFile pFile = LCell_GetFile(pCell);
    if(Assigned(pFile))
    {
        if(LCell_RunDRCEx00(pCell, NULL, LFile_LocUtoIntU(pFile, 100),
            NULL, LTRUE, LFALSE) == LStatusOK)
        {
            // More Processing
            // ...
        }
        // endif(Assigned(pFile))
    }
    // endif(Assigned(pCell))
}
```

Version

Available in L-Edit 8.2 and later versions.

**Note:** Note that this function is obsolete in L-Edit V10 and later.

See Also

LCell_RunDRCCommandFile

LStatus LCell_RunDRCCommandFile( LCell pCell, const char* szCommandFile, const LRect* pDRCArea, unsigned int* pnNumErrors);

Description

LCell_RunDRCCommandFile runs HiPer using a Calibre, Assura or Dracula DRC command file. It runs on the entire cell or a specified area of the cell.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value with possible values:

LBadCell — pCell is NULL
LSystemError — DRC didn't run due to some problem (parsing error, crash, etc).
LUserAbort — The user canceled DRC while it was running.
LCopyProtViolation — HiPer could not get a license.

Parameters

pCell Pointer to the cell.

CommandFile String with the path and filename of the HiPer command file. If this is an empty string "", then it will run the standard DRC rules.

pDRCArea Rectangular area to perform DRC on. If pDRCArea is NULL, then DRC is performed on the entire cell.

pnNumErrors This should be the address to a integer variable where the number of DRC errors found will be stored. It can be NULL.

Example

LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    unsigned int nNumOfErrors = 0;
    if(LCell_RunDRCCommandFile(pCell, "C:\\TSMC\\TSMC_HiPer.cal", NULL, &nNumOfErrors) == LStatusOK)
    {
        // More Processing
        // ...
    }
}

Example

Available in L-Edit 11.0 and later versions.
LCell_ClearUndoLists

LStatus LCell_ClearUndoLists( LCell cell );

Description

Clears the undo list for a cell. After calling this function the user can no longer undo his previous changes.

Always call LCell_ClearUndoLists whenever the macro modifies the design in such a way that the user's prior actions should not be undone. For example, if after manually removing the last polygon from the layer Active the user runs a macro that removes the layer Active from the design, then re-introducing of the removed polygon on a non-existing layer should not be permitted. This is why the function LCell_ClearUndoLists is called internally from LLayer_Delete to prevent the user from re-introducing invalid features by clicking Undo. It is a good practice to call LCell_ClearUndoLists at the end of a complex macro.

Return Values

Returns LStatusOK if successful, LBadCell in case of failure.

Parameters

cell

The cell for which the Undo should be cleared.

Example

LCell_ClearUndoLists( pCell );

Version

Available in L-Edit 8.4 and later versions.

See Also

“LLayer_Delete” (page 1531)
LCell_GetParameter

Note: This function is deprecated as of v15.00.

unsigned long LCell_GetParameter( LCell cell, const char *lpszParamName);

Description

Gets a T-Cell parameter value for the cell and parameter specified. If a T-Cell is specified, this function gets the default value of the parameter. If a cell generated from a T-Cell is specified, this function gets the parameter value used to generate the cell.

L-Edit automatically generates a call to LCell_GetParameter when it creates a T-Cell generator code template. Normally, you will not need to add any calls to this function.

Return Values

Returns the value of the requested parameter.

Parameters

- **cell** Pointer to the T-Cell or auto-generated cell
- **lpszParamName** Name of the T-Cell parameter.

Version

Available in L-Edit 9.0 and later versions.

See Also

The data type “LTCellParameterType” (page 1796) replaces this function. See also “LCell” (page 1700), and “LInstance_Generate” (page 1342).
LCell_GetTCellPreviousValue

unsigned long LCell_GetTCellPreviousValue( LCell cell, const char* cszParameterName, char* szBuffer, int nBufferLength );

Description

LCell_GetTCellPreviousValue is used to obtain the previous value of a T-Cell parameter, which is the value the last time the T-Cell was created or modified. If the T-Cell is newly created, an empty string is returned.

This function can be used in conjunction with LCell_SetTCellDefaultValue to implement callbacks, in which a T-Cell can validate and modify parameters that are passed to it.

Return Values

Returns the value of the requested parameter if successful, or an empty string if the T-Cell is new.

Parameters

cell Pointer to the T-Cell or auto-generated cell
cszParameterName Name of the T-Cell parameter.
szBuffer Previous parameter value.
nBufferLength Size of the previous parameter.

Example

The following utility function can be used to determine whether a particular parameter has changed from the previous invocation of the T-Cell:

```c
int HasChanged( LCell pCell, char *param_name )
{
    char old_val[1024], new_val[1024];
    LCell_GetTCellPreviousValue(pCell, param_name, old_val, sizeof(old_val));
    LCell_GetTCellDefaultValue(pCell, param_name, new_val, sizeof(new_val));
    return strcmp( old_val, new_val );
}
```

Then, in the main T-Cell code body, we can use this information to modify other parameters. For example, suppose we had a resistor that was parameterized by R, L and W. We want these three to be consistent, and we need to modify one of them to enforce this consistancy. One possible solution would be:

```c
// update new parameters accordingly
char new_val[1024];
if ( HasChanged(cellCurrent, "L") && HasChanged(cellCurrent, "W") )
{
    R = L / W * resistivity;
    sprintf( new_val, "%g", R );
}
```
LCell_SetTCellDefaultValue(cellCurrent, "R", new_val);
}
else if ( HasChanged(cellCurrent, "L") )
{
    W = L * resistivity / R;
sprintf( new_val, "%g", W );
    LCell_SetTCellDefaultValue(cellCurrent, "W", new_val);
}
else
{
    L = W * R / resistivity;
sprintf( new_val, "%g", L );
    LCell_SetTCellDefaultValue(cellCurrent, "L", new_val);
}

Version

Available in L-Edit 12.20 and later versions.

See Also

LCell_GetTCellDefaultValue
**LCell_SetShowInLists**

```c
LStatus LCell_SetShowInLists( LCell cell, LBoolean show );
```

**Description**

Sets the **Show in Lists** cell attribute. When this attribute is TRUE, the cell is always listed in the Design Navigator, **Cell > Open**, and **Cell > Instance** dialogs. When FALSE, the cell is hidden from lists. You can show hidden cells by selecting **Show All Cells** in the Design Navigator, **Cell > Open**, or **Cell > Instance** dialogs.

**Return Values**

*LStatusOK* if successful. If an error occurs, *LStatus* contains the error value.

**Parameters**

- **cell**: Pointer to the cell.
- **show**: **Show in Lists** state. Possible values are:
  - *LTRUE*: Always include the cell in lists.
  - *LFALSE*: Omit the cell from cell lists, except when the **Show All Cells** option is selected in L-Edit.

**Version**

Available in L-Edit 9.0 and later versions.

**See Also**

**LCell_GetShowInLists**

```c
LBoolean LCell_GetShowInLists( LCell cell );
```

**Description**

Gets the state of the **Show in Lists** cell attribute. When this attribute is TRUE, the cell is always listed in the Design Navigator, **Cell > Open**, and **Cell > Instance** dialogs. When FALSE, the cell is hidden from lists. You can show hidden cells by selecting **Show All Cells** in the Design Navigator, **Cell > Open**, or **Cell > Instance** dialogs.

**Return Values**

Returns the state of the **Show in Lists** attribute, with possible values:

- **LTRUE**—Always include the cell in lists.
- **LFALSE**—Omit the cell from cell lists, except when the **Show All Cells** option is selected in L-Edit.

**Version**

Available in L-Edit 9.0 and later versions.

**See Also**

“LCell_SetShowInLists” (page 1312), “LBoolean” (page 1697), “LCell” (page 1700)
LCell_CalcMBB

    void LCell_CalcMBB( LCell pCell );
LCell_AddMarker

LMarker LCell_AddMarker(LCell pCell, const char* cszText, int nNumOfVertices, LBoolean bPolygon, const LPoint* cpnVertices, const LMarkerParam* cpParam);
LCell_RemoveMarker

```c
void LCell_RemoveMarker( LCell pCell, LMarker MarkerHandle );
```
LCell_RemoveAllMarkers

```c
void LCell_RemoveAllMarkers( LCell pCell );
```
**LCell_BooleanOperation**

```c
LStatus LCell_BooleanOperation( LCell pCell, LBooleanOperation nOp,
    LCoord Amount, LObject* ArrayOfObjectsA, unsigned int nNumOfObjectsA,
    LObject* ArrayOfObjectsB, unsigned int nNumOfObjectsB, LLayer pResultLayer, LBoolean bDeleteInputs );
```

**Description**

Performs Boolean operations on a group of objects. You must perform AND, XOR and SUBTRACT on two groups of objects, all other Boolean operations operate on just one group of objects. (This is similar to the **Draw > Boolean/Grow Operations** command in L-Edit.)

**Return Values**

Returns the state of the **Show in Lists** attribute, with possible values:

- **LTRUE**
  - Always include the cell in lists.
- **LFALSE**
  - Omit the cell from cell lists, except when the Show All Cells option is selected in L-Edit.

**Parameters**

- **pCell**
  - Cell in which to put the resultant objects.
- **nOp**
- **nAmount**
  - The amount by which to grow or shrink the object.
- **ArrayOfObjectsA**
  - Array of objects in group A.
- **nNumOfObjectsA**
  - Number of objects in group A.
- **ArrayOfObjectsB**
  - Array of objects in group B.
- **nNumOfObjectsB**
  - Number of objects in group B.
- **pResultLayer**
  - Layer on which to put the resultant objects.
- **bDeleteInput**
  - Deletes the original objects.

**Example 1**

```c
LCell pCell = LCell_GetVisible();
LFile pTDBFile= LCell_GetFile(pCell);
int nNumOfM1Boxes = 0;
LLayer pM1Layer = LLayer_Find(pTDBFile, "Metal1");
LObject pObject = NULL;
```
// Count the number of boxes on Metal 1
for(pObject = LObject_GetList(pCell, pM1Layer);
    Assigned(pObject);
    pObject = LObject_GetNext(pObject))
{
    if(LObject_GetShape(pObject) == LBox)
        nNumOfM1Boxes++;
}

// Allocate the memory for the object array.
LObject* paObjects = (LObject*)malloc(nNumOfM1Boxes*sizeof(LObject));

int nIndex = 0;
for(pObject = LObject_GetList(pCell, pM1Layer);
    Assigned(pObject);
    pObject = LObject_GetNext(pObject))
{
    if(LObject_GetShape(pObject) == LBox)
        paObjects[nIndex++] = pObject;
}

// Grow all boxes on M1 by 1 micron.
LCell_BooleanOperation(pCell, LBoolOp_GROW, 1000, paObjects, nNumOfM1Boxes,
                        NULL, 0, pM1Layer, LTRUE);

// Free the memory for the object array.
free(paObjects);

Example 2

LCell pCell = LCell_GetVisible();
LFile pFile = LCell_GetFile(pCell);

int nNumElements = 0;
for( LObject pObjects=LObject_GetList(pCell,LLayer_Find(pFile,"L1"));Assigned(pObjects );pObjects=LObject_GetNext(pObjects))
    {nNumElements++;
    for( LObject pObjects=LObject_GetList(pCell,LLayer_Find(pFile,"L2"));Assigned(pObjects );pObjects=LObject_GetNext(pObjects))
        {nNumElements++;
        LObject* paObjects = (LObject*)malloc(nNumElements*sizeof(LObject));
        int nIndex = 0;
        for( LObject pObjects=LObject_GetList(pCell,LLayer_Find(pFile,"L1"));Assigned(pObjects );pObjects=LObject_GetNext(pObjects))
            { paObjects[nIndex++] = pObjects; }
        for( LObject pObjects=LObject_GetList(pCell,LLayer_Find(pFile,"L2"));Assigned(pObjects );pObjects=LObject_GetNext(pObjects))
            { paObjects[nIndex++] = pObjects; }
        LCell_BooleanOperation(pCell,LBoolOp_OR,NULL,paObjects,nIndex,NULL, 0,
                               LLayer_Find(pFile,"L3"),LFALSE);
Version

Available in L-Edit 10.0 and later versions.
LCell_Slice

LStatus LCell_Slice(LCell pCell, LObject* pObjectArray,
                    unsigned int nNumOfObjects, const LPoint *cpPoint1, const LPoint *cpPoint2);

LStatus LCell_Slice(LCell pCell, LObject* pObjectArray,
                    unsigned int nNumOfObjects, const LPoint *cpPoint1, const LPoint *cpPoint2);
LCell_SetBasePoint

LStatus LCell_SetBasePoint(LCell pCell, LCoord lX, LCoord lY);
LCell_GetBasePoint

LPoint LCell_GetBasePoint( LCell pCell);
**Instance Functions**

An instance is a reference to a cell (the instanced cell) from within another cell (the instancing cell). Each instancing cell maintains a list of instances in an `LInstance` data structure.

Instance functions allow the user to manipulate an instance of a cell.

```
“LInstance_New_Ex99” (page 1326)        “LInstance_Delete” (page 1327)
“LInstance_Set_Ex99” (page 1329)        “LInstance_GetMbb” (page 1341)
“LInstance_Find” (page 1330)           “LInstance_FindNext” (page 1331)
“LInstance_GetList” (page 1332)         “LInstance_GetNext” (page 1333)
“LInstance_GetName” (page 1334)         “LInstance_SetName” (page 1335)
“LInstance_GetTransform_Ex99” (page 1338) “LInstance_GetCell” (page 1336)
“LInstance_GetRepeatCount” (page 1339)  “LInstance_GetDelta” (page 1340)
“LInstance_Generate” (page 1342)        “LInstance_GenerateV” (page 1343)
```

Obsolete:

```
“LInstance_New” (page 1325)        “LInstance_Set” (page 1328)
“LInstance_GetTransform” (page 1337)
```

```
```
LInstance_New

LInstance LInstance_New( LCell cell, LCell instance_cell, LTransform transform, LPoint repeat_cnt, LPoint delta );

Description

Creates a new instance or array of instances in the specified cell. (An array is a geometrically regular
two-dimensional arrangement of instances of a single cell.)

The array repeat count specified in \texttt{repeat\_cnt} and array spacing offset specified in \texttt{delta} specify how
an array of instances will be created.

The parameters \texttt{cell} and \texttt{instance\_cell} must be in the same file.

Return Values

Returns a pointer to the newly created instance or array if successful; otherwise returns NULL.

Parameters

- \texttt{cell} Instancing cell.
- \texttt{instance\_cell} Instanced cell.
- \texttt{transform} Translation and rotation of the new instance.
- \texttt{repeat\_cnt} Ordered pair specifying the dimensions of the array. The first number
  in the pair specifies rows; the second number specified columns.
  Minimum value is 1,1.
- \texttt{delta} Ordered pair specifying the spacing offset of the array.

\textbf{Note:} Note that this function is obsolete in L-Edit V10 and later.

See Also

“\texttt{LObject}” (page 1769), “\texttt{LTransform}” (page 1801), “\texttt{Instance Functions}” (page 1324)
# LInstance_New_Ex99

```c
LInstance LInstance_New_Ex99( LCell cell, LCell instance_cell, LTransform_Ex99 transform, LPoint repeat_cnt, LPoint delta );
```

## Description

Creates a new instance or array of instances in the specified cell. (An array is a geometrically regular two-dimensional arrangement of instances of a single cell.)

The array repeat count specified in `repeat_cnt` and array spacing offset specified in `delta` specify how an array of instances will be created.

The parameters `cell` and `instance_cell` must be in the same file.

## Return Values

Returns a pointer to the newly created instance or array if successful; otherwise returns NULL.

## Parameters

- **cell** Instancing cell.
- **instance_cell** Instanced cell.
- **transform** Translation and rotation of the new instance.
- **repeat_cnt** Ordered pair specifying the dimensions of the array. The first number in the pair specifies rows; the second number specified columns. Minimum value is 1,1.
- **delta** Ordered pair specifying the spacing offset of the array.

## See Also

**LInstance_Delete**

```c
LStatus LInstance_Delete( LCell cell, LInstance instance );
```

**Description**

Deletes the specified instance from the specified cell.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

**Parameters**

- `cell` Instancing cell.
- `instance` Instance to be deleted.

**See Also**

“L.Status” (page 1792), “L.Object” (page 1769), “Instance Functions” (page 1324)
**LInstance_Set**

```c
LStatus LInstance_Set( LCell cell, LInstance instance,
                      LTransform transform, LPoint repeat_cnt, LPoint delta );
```

**Description**

Modifies the specified instance or array of instances in the specified cell with new values for translation, rotation, dimension, and offset.

**Return Values**

Returns **LStatusOK** if successful. If an error occurs, **LStatus** contains the error value.

**Parameters**

- **cell**: Instancing cell.
- **instance**: Instance to be modified.
- **transform**: Translation, rotation, and magnification of the instance.
- **repeat_cnt**: Dimensions of the array.
- **delta**: Spacing offset of the array.

**Note:** Note that this function is obsolete in L-Edit V10 and later.

**See Also**

“**LStatus**” (page 1792), “**LObject**” (page 1769), “**LTransform**” (page 1801),
“**Instance Functions**” (page 1324)
LInstance_Set_Ex99

\[
\text{LStatus } \text{LInstance_Set_Ex99( LCell } \text{ cell, LInstance } \text{ instance, LTransform_Ex99 } \text{ transform, LPoint } \text{ repeat_cnt, LPoint } \text{ delta );}
\]

Description

Modifies the specified instance or array of instances in the specified cell with new values for translation, rotation, dimension, and offset.

Return Values

Returns \text{LStatusOK} if successful. If an error occurs, \text{LStatus} contains the error value.

Parameters

- \text{cell} Instancing cell.
- \text{instance} Instance to be modified.
- \text{transform} Translation, rotation, and magnification of the instance.
- \text{repeat_cnt} Dimensions of the array.
- \text{delta} Spacing offset of the array.

See Also

“\text{LStatus}” (page 1792), “\text{LObject}” (page 1769), “\text{LTransform}” (page 1801), “\text{Instance Functions}” (page 1324)
**LInstance_Find**

\[ \text{LInstance LInstance\_Find( LCell } pCell, \text{ const char* szName } ); \]

**Description**

Searches for an instance of the specified name in the specified cell.

**Return Values**

Returns a pointer to the instance if successful; otherwise returns NULL.

**Parameters**

- \( pCell \) Instancing cell to search for instances.
- \( szName \) Name of instance to search for.

**See Also**

“LInstance” (page 1752), “LCell” (page 1700), “Instance Functions” (page 1324)
LInstance_FindNext

LInstance LInstance_FindNext( LInstance instance, const char* name );

Description

Continues the search for an instance of the specified name (proceeding from the last such instance).

Return Values

Returns a pointer to the next instance if successful; otherwise returns NULL.

Parameters

instance
Most recently found instance.

name
Name of instance to search for.

See Also

“LObject” (page 1769), “Instance Functions” (page 1324)
LInstance_GetList

```c
LInstance LInstance_GetList( LCell cell );
```

Description

Gets the first instance in the specified cell’s list of instances.

Return Values

Returns a pointer to the instance list if successful; otherwise returns NULL.

Parameters

- `cell` Instancing cell.

See Also

**LInstance_GetNext**

```c
LInstance LInstance_GetNext( LInstance instance );
```

**Description**

Gets the next instance after the specified instance in the current cell’s list of instances.

**Return Values**

Returns a pointer to the next instance if successful; otherwise returns NULL.

**Parameters**

`instance` Specified instance.

**See Also**

LInstance_GetName

```c
char* LInstance_GetName( LInstance instance, char* name, const int maxlen );
```

**Description**

Gets the name of the specified instance as a string (up to a maximum string length).

**Return Values**

Returns a pointer to the instance name buffer if successful; otherwise returns NULL.

**Parameters**

- `instance` Instance whose name is to be retrieved.
- `name` String (buffer) containing the name text.
- `maxlen` Maximum length allowed for `name`.

**See Also**

“LObject” (page 1769), “LFile_GetName” (page 1208), “LCell_GetName” (page 1271), “Instance Functions” (page 1324)
**LInstance_SetName**

```c
LStatus LInstance_SetName( LCell cell, LInstance instance, char* name );
```

**Description**

Sets the name of the specified instance in the specified cell.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

**Parameters**

- **cell** Cell containing the instance.
- **instance** Instance to be (re)named.
- **name** New name of the instance.

**See Also**

“L.Status” (page 1792), “LObject” (page 1769), “Instance Functions” (page 1324)
LInstance_GetCell

```c
LCell LInstance_GetCell( LInstance instance );
```

Description

Gets the parent (instanced) cell of the specified instance.

Return Values

Returns a pointer to the parent cell if successful; otherwise returns NULL.

Parameter

- `instance`: Specified instance.

See Also

- “LObject” (page 1769), “Instance Functions” (page 1324)
**LInstance_GetTransform**

```c
LTransform LInstance_GetTransform( LInstance instance );
```

**Description**

Gets the transformation of the specified instance.

**Return Values**

Returns the translation, magnification, and rotation of the specified instance; returns a zero transform on error.

**Parameters**

- `instance` Specify instance.

---

**Note:**

Note that this function is obsolete in L-Edit V10 and later.

**See Also**

“LTransform” (page 1801), “LObject” (page 1769), “Instance Functions” (page 1324)
LInstance_GetTransform_Ex99

LTransform_Ex99 LInstance_GetTransform_Ex99( LInstance instance );

Description

Gets the transformation of the specified instance.

Return Values

Returns the translation, magnification, and rotation of the specified instance; returns a zero transform on error.

Parameters

instance

Specified instance.

See Also

LInstance_GetRepeatCount

LPoint LInstance_GetRepeatCount( LInstance instance );

Description

Gets the repeat count of an instance.

Return Values

Returns the array dimensionality of the specified instance as an ordered pair, or (1,1) for non-array instances; returns (0,0) on error.

Parameters

instance

Specified instance.

See Also

“LTransform” (page 1801), “LObject” (page 1769), “Instance Functions” (page 1324)
LInstance_GetDelta

\[ \text{LPoint } \text{LInstance}_\text{GetDelta}( \text{LInstance } \text{instance} ); \]

**Description**

Gets the array spacing of the specified instance as an ordered pair.

**Return Values**

Returns the array spacing of the specified instance as an ordered pair; returns \((0,0)\) on error

**Parameters**

\[ \text{instance} \quad \text{Specified instance.} \]

**See Also**

“LTransform” (page 1801), “LObject” (page 1769), “Instance Functions” (page 1324)
LnInstance_GetMbb

\[ \text{LRect } \text{LnInstance_GetMbb( LInstance } \text{instance } ); \]

Description

Gets the Mbb of an instance.

Return Values

Returns the coordinates of the rectangle representing the minimum bounding box (Mbb) of the specified instance; on error returns a rectangle whose coordinates are all zeros.

Parameters

\text{instance } \quad \text{Specified instance.}

See Also

“LTransform” (page 1801), “LObject” (page 1769), “Instance Functions” (page 1324)
Limstance_Generate

Limstance_Limstance_Generate(LCell cell, LCell TCell, params...);  

Description

Creates an instance of a T-Cell in the specified cell, passing parameters to the T-Cell generator code.

Return Values

Returns a pointer to the newly created instance if successful; otherwise, returns NULL.

Parameters

- cell: Pointer to the cell in which to place the T-Cell instance.
- TCell: Pointer to the T-Cell generator to be instanced.
- params...: T-Cell parameters. Parameters are passed as pairs of strings (i.e., const char*), each representing a parameter name and corresponding value.

If you pass fewer parameters than are defined for the instanced T-Cell, you must end the list with a NULL or empty string for the next parameter name.

Version

Available in L-Edit 9.0 and later versions.

See Also

“LCell” (page 1700), “LImstance” (page 1752), “LImstance_GenerateV” (page 1343)
LInstance_GenerateV

LInstance LInstance_GenerateV( LCell cell, LCell instance_cell, char **argList );

Description

Creates an instance of a cell in the specified cell.

Return Values

Returns a pointer to the newly created instance if successful; otherwise, returns NULL.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>Pointer to the cell in which to place the cell instance.</td>
</tr>
<tr>
<td>LCell</td>
<td>Pointer to the cell to be instanced.</td>
</tr>
</tbody>
</table>

Example 1

/* This example creates an instance of cell DecoderGen, passing two parameter values to the T-Cell generator. */
char* params[5]; /* array of pointers to character strings */

/* Initialize parameter variables*/
int Outputs = 4;
int DecoderBits = 2;

/* Create array of parameter names and values */
/* parameter 1, name and value */
params[0] = "DecoderBits"; params[1] = LFormat("%d", DecoderBits);

/* parameter 2, name and value */

/* End the parameter list with NULL */
params[4] = NULL;

LInstance_GenerateV(MyCell, DecoderGen, params);

Example 2

int add_contact( LCell cell, char * l )
{
    char *params[3];
    LCell tcell;
    LPoint here;
    LFile file;
    LTransform_Ex99 xform;
    LInstance inst;
    LPoint one;
    params[0] = "bottom";
params[1] = 1;
params[2] = NULL;
file = LCell_GetFile(cell);
tcell = LCell_Find(file, "contact");
here = LCursor_GetSnappedPosition();
if (tcell == NULL) {
    LDialog_AlertBox("failed to find contact");
    return 1;
}
inst = LInstance_GenerateV(cell, tcell, params);
xform = LInstance_GetTransform_Ex99(inst);
xform.translation = here;
one.y = one.x = 1.0;
LInstance_Set_Ex99(cell, inst, xform, one, one);
**Entity Functions**

These functions control properties for **LFile**, **LCell**, **LLayer**, or **LObject** (**LBox**, **LPolygon**, **LWire**, **LPort**, **LCircle**, and **LInstance**) entities. Note that the **LFile**, **LCell**, **LLayer**, and **LObject** functions must be cast to **LEntity** for use with the Entity Functions.

- "LEntity_PropertyExists" (page 1346)
- "LEntity_GetPropertyValueType" (page 1347)
- "LEntity_GetPropertyValueSize" (page 1348)
- "LEntity_GetPropertyValue" (page 1349)
- "LEntity_AssignProperty" (page 1350)
- "LEntity_AssignBlobProperty" (page 1351)
- "LEntity_DeleteProperty" (page 1352)
- "LEntity_DeleteAllProperties" (page 1353)
- "LEntity_CopyAllProperties" (page 1354)
- "LEntity_GetFirstProperty" (page 1355)
- "LEntity_GetNextProperty" (page 1356)
- "LEntity_SetCurrentProperty" (page 1357)
- "LEntity_BrowseProperties" (page 1358)
- "LEntity_ValidPropertyNameToString" (page 1365)
- "LEntity_StringToValidPropertyName" (page 1363)
- "LEntity_ReadPropertiesFromFile" (page 1361)
- "LEntity_WritePropertiesToFile" (page 1367)
- "LEntity_LoadBlobProperty" (page 1359)
- "LEntity_SaveBlobProperty" (page 1360)
- "LEntity_StoreAsCompressedBlob" (page 1369)
- "LEntity_DecompressBlobToFile" (page 1370)
LEntity_PropertyExists

```c
LStatus LEntity_PropertyExists( const LEntity entity, const char* name );
```

Description

Determines whether a property exists.

Return Values

Returns L.StatusOK if the property is found. If an error occurs, L.Status contains the error value.

Parameters

- `entity` A pointer to an LEntity.
- `name` The path of the property.
LEntity_GetPropertyType

```c
LStatus LEntity_GetPropertyType ( const LEntity entity, const char* name, LPropertyType *type );
```

**Description**

Retrieves the property’s type.

**Return Values**

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `entity` A pointer to an LEntity.
- `name` The path of the property.
- `type` A pointer to the property type.
LEntity_GetPropertyValueSize

```c
unsigned int LEntity_GetPropertyValueSize( const LEntity entity, const char* name );
```

**Description**

Retrieves the size of a property’s value.

**Return Values**

Returns the size of the value if the property is found and it has a value; otherwise, returns zero.

**Parameters**

- `entity` A pointer to an LEntity.
- `name` The path of the property.
LEntity_GetPropertyValue

```c
LStatus LEntity_GetPropertyValue (const LEntity entity, const char* name, void* value, unsigned int max_size)
```

**Description**

Retrieves a property’s value.

**Return Values**

Returns LStatusOK if successful. If an error occurs, LStatus contains the error.

**Parameters**

- **entity**
  A pointer to an LEntity.
- **name**
  The path of the property.
- **value**
  A pointer to the value.
- **max_size**
  The maximum size of the buffer pointed to by the value.
LEntity_AssignProperty

LEntity_AssignProperty( LEntity entity, const char* name, LPropertyType type, const void* value );

Description

Creates a new property and assigns a type and value, or changes or removes the value of an existing property. An existing property’s type cannot be changed.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>A pointer to an LEntity.</td>
</tr>
<tr>
<td>name</td>
<td>The path of the property.</td>
</tr>
<tr>
<td>value</td>
<td>A pointer to the value. If NULL, no value is assigned to a new property, or the current value of an existing property is removed.</td>
</tr>
<tr>
<td>type</td>
<td>The property’s type.</td>
</tr>
</tbody>
</table>
**LEntity_AssignBlobProperty**

```c
LStatus LEntity_AssignBlobProperty( LEntity entity, const char* name,
const void* value, unsigned int size );
```

**Description**

Creates a new blob property and a value, changes or removes the value of an existing property.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error.

**Parameters**

- **entity**
  A pointer to an LEntity.

- **name**
  The path of the property.

- **value**
  A pointer to the value. If NULL, no value is assigned to a new property, or the current value of an existing property is removed.

- **size**
  The size of the value.
**LEntity_DeleteProperty**

```c
LStatus LEntity_DeleteProperty( LEntity entity, const char* name );
```

**Description**

Deletes a property.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error.

**Parameters**

- **entity**
  - A pointer to an LEntity.
- **name**
  - The path of the property.
LEntity_DeleteAllProperties

```c
LStatus LEntity_DeleteAllProperties( LEntity entity );
```

**Description**

Deletes all properties on an entity.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error.

**Parameters**

- `entity`  
  A pointer to an LEntity.
- `name`  
  The path of the property.
**LEntity_CopyAllProperties**

```c
LStatus LEntity_CopyAllProperties( LEntity target_entity,
                                    const LEntity source_entity );
```

**Description**

Copies all of one entity’s properties to another entity overwriting the other entity’s properties.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error.

**Parameters**

- `target_entity` A pointer to the target entity.
- `source_entity` A pointer to the source entity.
**LEntity_GetFirstProperty**

```c
const char* LEntity_GetFirstProperty( const LEntity entity );
```

**Description**

Retrieves the first property of an entity.

**Return Values**

Returns the name first property on an entity or NULL if the entity has no properties.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error.

**Parameters**

- `entity` A pointer to an LEntity.
LEntity_GetNextProperty

```c
const char* LEntity_GetNextProperty( void );
```

**Description**

Retrieves the next property of an entity or NULL if there are no more properties on the entity.

**Note:**

If the current property is deleted or renamed, the next call will return NULL, unless an appropriate call to `LEntity_SetCurrentProperty` is made first.

**Return Values**

Returns the name of the next property on an entity or NULL if the entity has no more properties.

**Parameters**

- `entity` A pointer to an LEntity.
LEntity_SetCurrentProperty

    void LEntity_SetCurrentProperty( const char* name );

Description

Sets the name of the current property in the traversal of the property tree.

Return Values

Returns the name of the path of the current property on an entity.

Parameters

    name            The full path of the property.
**LEntity_BrowseProperties**

```cpp
LStatus LEntity_BrowseProperties( LEntity entity );
```

**Description**

Invokes the standard property browser.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error.

**Parameters**

- `entity` The entity on which to invoke the browser.
LEntity_LoadBlobProperty

```c
LStatus LEntity_LoadBlobProperty(LEntity entity, const char* name, const char* file_name)
```

**Description**

Sets a blob property from a file.

**Return Values**

Returns the name of the next property on an entity or NULL if the entity has no more properties.

**Parameters**

- `entity` A pointer to an LEntity.
- `name` The path of the property
- `file_name` The name of the file containing the value.
LEntity_SaveBlobProperty

```c
LStatus LEntity_SaveBlobProperty( const LEntity entity, const char* name, void* value, const char* file_name)
```

**Description**

Saves a blob property’s value to a file.

**Return Values**

Returns the name of the next property on an entity or NULL if the entity has no more properties.

**Parameters**

- **entity**: A pointer to an LEntity.
- **name**: The path of the property
- **file_name**: The name of the file containing the value.
LEntity_ReadPropertiesFromFile

LEntity_ReadPropertiesFromFile( LEntity pEntity, const char *szPath, const char* szFilename)

Description

Reads a text file with property information in TTX format and assigns the properties to the indicated property subtree on the specified entity (File, Cell, Layer, or Object).

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value with possible values:

LBadObject — if pEntity is NULL.
LBadParameters — if szFilename is NULL.
LCreateError — if the file szFilename could not be opened for reading.
LSyserror — if tpropprs.dll could not be loaded.
LPropertyConversionError — if an error occurred during the importing of the properties from a file.

Parameters

pEntity Entity that has the properties to write to the file.

szPath Path to the property subtree to write to the file. If szPath = NULL, then all properties are written to the file.

szFileName Export filename.

Example

LFile pFile = LFile_GetVisible();
if(Assigned(pFile)) =
{
    if(LEntity_ReadPropertiesFromFile((LEntity)pFile, "MyProperties", "MyProp.ttx") == LStatusOK)
    {
        // More Processing
        // ...
    }
} // endif(Assigned(pFile))

Version

Available in L-Edit 8.2 and later versions.
See Also

LEntity_StringToValidPropertyName

```c
LStatus LEntity_StringToValidPropertyName( const char* szString,
  char *szPropertyName, int iBufferSize );
```

**Description**

Converts a string to a valid property name by replacing all underscores (_) with double underscores (___) and all invalid characters to ___## where ## is the two digit ASCII number in hexadecimal for that character. A valid property name can consists of letters, numbers, spaces, and underscores.

**Return Values**

*LStatusOK* if successful. If an error occurs, *LStatus* contains the error value with possible values:

- *LBadParameters* — One or more of the following:
  - *szString* is NULL.
  - *szPropertyName* is NULL.
  - *iBufferSize* < 2.

**Parameters**

- **szString** String to convert.
- **szPropertyName** String buffer to store the result in.
- **iBufferSize** Size of the *szPropertyName* string buffer.

**Example**

```c
char szPropertyName[256];
if(LEntity_StringToValidPropertyName("Metal 1 & Metal 2 - DRC",
  szPropertyName, 255) == LStatusOK)
{
    // szPropertyName now equals "Metal 1 _038 Metal 2 _045 DRC"

    LFile pFile = LFile_GetVisible();
    if(Assigned(pFile))
    {
        int iData = 5;
        if(LEntity_AssignProperty((LEntity)pFile, szPropertyName,
            L_int, &iData) == LStatusOK)
            // More Processing
            // ...
    } // endif(Assigned(pFile))
} // endif(LEntity_StringToValidPropertyName("Metal 1 & Metal 2 - DRC",
  szPropertyName, 255) == LStatusOK)
```
Version

Available in L-Edit 8.2 and later versions.

See Also

LEntity_ValidPropertyNameToString

```
LEntity_ValidPropertyNameToString( const char* szPropertyName, char* szString, int iBufferSize );
```

**Description**

Converts a property name to a string by replacing all double underscores (___) with single underscores (_ _) and converting _## to the ASCII character whose number is ## in hexadecimal. A valid property name can consist of letters, numbers, spaces, and underscores. This function is used in conjunction with LEntity_StringToValidPropertyName to store strings with invalid characters as property names.

**Return Values**

- **LStatusOK** if successful. If an error occurs, **LStatus** contains the error value with possible values:
  - **LBadParameters** — one or more of the following:
    - `szPropertyName` is NULL.
    - `szString` is NULL.
    - `iBufferSize` < 2.

**Parameters**

- **szPropertyName** 
  Property name to convert.
- **szString** 
  String buffer to store the result in.
- **iBufferSize** 
  Size of the szString string buffer.

**Example**

```
LFile pFile = LFile_GetVisible();
if(Assigned(pFile))
{
    // Get the first property.
    const char* pszPropertyName = LEntity_GetFirstProperty((LEntity)pFile);

    char szString[256];
    if(LEntity_ValidPropertyNameToString(pszPropertyName, szString, 255) == LStatusOK)
    {
        // szString now equals "Metal 1 & Metal 2 - DRC"

        // More Processing
        // ...
    } // endif(LEntity_ValidPropertyNameToString(pszPropertyName, szString, 255) == LStatusOK)
} // endif(Assigned(pFile))
```
Version

Available in L-Edit 8.2 and later versions.

See Also

LEntity_WritePropertiesToFile

**LStatus** LEntity_WritePropertiesToFile(const LEntity pEntity, const char *szPath, const char* szFilename);

**Description**

Write properties from the indicated property subtree on the specified entity (File, Cell, Layer, or Object) to a text file in TTX format.

**Return Values**

*StatusOK* if successful. If an error occurs, **LStatus** contains the error value with possible values:

- **LBadObject**—if *pEntity* is NULL.
- **LBadParameters**—if *szFilename* is NULL.
- **LCREATEERROR**—if the file *szFilename* could not be created.
- **LSystemError**—if *tpropprs.dll* could not be loaded.
- **LPropertyConversionError**—if an error occurred during the exporting of the properties to a file.

**Parameters**

- **pEntity** Entity that has the properties to write to the file.
- **szPath** Path to the property subtree to write to the file. If *szPath* = NULL, then all properties are written to the file.
- **szFileName** Export filename.

**Example**

```c
LFile pFile = LFile_GetVisible();
if(Assigned(pFile))
{
    if(LEntity_WritePropertiesToFile((LEntity)pFile, "MyProperties", "MyProp.ttx") == LStatusOK)
    {
        // More Processing
        // ...
    }
} // endif(Assigned(pFile))
```

**Version**

Available in L-Edit 8.2 and later versions.
See Also

**LEntity_StoreAsCompressedBlob**

```c
LStatus LEntity_StoreAsCompressedBlob( LEntity pEntity, const char *szFileName,
                                           const char* szBlobPropName, const char *szUncompressedSizePropName );
```
LEntity-DecompressBlobToFile

**LStatus** LEntity_DecompressBlobToFile( **LEntity** pEntity, const char *szFileName, const char* szBlobPropName, const char *szUncompressedSizePropName );
Object Functions

Object functions allow the user to manipulate an object in a cell. The categories of object functions are:

- "Vertex Functions" (page 1396)
- "Box Functions" (page 1411)
- "Circle Functions" (page 1415)
- "Torus and Pie Functions" (page 1421)
- "Wire Functions" (page 1432)
- "Polygon Functions" (page 1445)
- "Port Functions" (page 1452)

Object functions do not apply to ports (see “Port Functions” on page 1452) or instances (see “Instance Functions” on page 1324).
Object Functions

General object functions include:

- `LObject_GetList` (page 1374)
- `LObject_GetNext` (page 1375)
- `LObject_GetShape` (page 1380)
- `LObject_GetLayer` (page 1385)
- `LObject_GetGeometry` (page 1381)
- `LObject_GetVertexList` (page 1382)
- `LObject_GetLayer` (page 1385)
- `LObject_ChangeLayer` (page 1389)
- `LObject_GetGeometry` (page 1381)
- `LObject_GetInstance` (page 1378)
- `LObject_GetVertexList` (page 1382)
- `LObject_Perimeter` (page 1384)
- `LObject_GetGDSIIDataType` (page 1386)
- `LObject_Delete` (page 1373)
- `LObject_Transform` (page 1376)
- `LObject_SetGDSIIDataType` (page 1387)
- `LObject_Copy` (page 1392)
- `LObject_SnapToMfgGrid` (page 1394)
- `LObject_SnapToGrid` (page 1393)
- `LObject_Area` (page 1383)
- `LObject_DistanceToPoint` (page 1395)
- `LObject_GetMbb` (page 1379)
- `LObject_GetMbb` (page 1379)

Obsolete

- `LObject_Transform` (page 1376)
**LObject_Delete**

```
LStatus LObject_Delete( LCell cell, LObject object );
```

**Description**

Removes object from cell.

**Return Values**

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- **cell**
  Cell containing the object to be deleted.
- **object**
  Object to be deleted.

**See Also**

“LStatus” (page 1792), “LObject” (page 1769), “Object Functions” (page 1371)
LObject_GetList

LObject LObject_GetList( LCell cell, LLayer layer );

Description

Gets a list of objects in the specified cell on the specified layer.

Return Values

Returns a pointer to the head object in the current object list if successful; otherwise returns NULL.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>Specified cell.</td>
</tr>
<tr>
<td>layer</td>
<td>Layer on which objects are drawn.</td>
</tr>
</tbody>
</table>

See Also

LObject_GetNext

LObject LObject_GetNext( LObject object );

Description

Gets the next object which follows the specified object.

Return Values

Returns a pointer to the next object in the object list if successful; otherwise returns NULL.

Parameters

object Specified object.

See Also

“LObject” (page 1769), “Object Functions” (page 1371)
LObject_Transform

void LObject_Transform( LObject object, LTransform transform );

Description

Transforms an object.

Parameters

object Specified object.
transform Specified transform.

Note: Note that this function is obsolete in L-Edit V10 and later.

See Also

“LObject” (page 1769), “LTransform” (page 1801), “Object Functions” (page 1371)
LObject_Transform_Ex99

```c
void LObject_Transform_Ex99( Object object, LTransform_Ex99 transform );
```

**Description**

Transforms an object.

**Parameters**

- `object` Specified object.
- `transform` Specified transform.

**See Also**

Chapter 34: UPI Functions Reference

LObject_GetInstance

**LInstance** LObject_GetInstance( **LObject** pObject )

**Description**

Converts an **LObject** to an LInstance if possible.

**Return Values**

If pObject is a LInstance then a LInstance, NULL otherwise.

**Parameters**

pObject Object to convert.

**See Also**

**LObject_GetMbb**

```c
LRect LObject_GetMbb( LObject object );
```

**Description**

Gets the minimum bounding box of an object.

**Return Values**

Returns the coordinates of the Mbb rectangle if successful; otherwise returns a rectangle whose coordinates are all zeros.

**Parameters**

- `object` Specified object.

**See Also**

“LObject” (page 1769), “Object Functions” (page 1371)
**LObject_GetShape**

**LShapeType**  *LObject_GetShape( LObject pObject )*

**Description**

Gets the shape of an object.

**Return Values**

Returns the shape of object as an **LShapeType** enum. Possible values include box, circle, wire, polygon, torus, pie wedge, instance, port, ruler, or other.

**Parameters**

- **pObject**  Specified object.

**See Also**

“**LShapeType**” (page 1790), “**LObject**” (page 1769), “**Object Functions**” (page 1371)
**LObject_GetGeometry**

```c
LGeomType LObject_GetGeometry( LObject object );
```

**Description**

Gets the geometry specifications of an object.

**Return Values**

Returns the geometric constraint of object as an **LGeomType** enum. Geometry types include orthogonal, 45-degree angle, and all-angle.

**Parameters**

- **object**
  
  Specified object.

**See Also**

- “[LTransform](page 1801)”
- “[LObject](page 1769)”
- “[Object Functions](page 1371)”
**LObject_GetVertexList**

```c
LVertex LObject_GetVertexList( LObject object );
```

**Description**

Retrieves the first vertex of an object. This works only on `LPolygon` and `LWire`.

**Return Values**

Returns a pointer to the first vertex in a polygon or wire object’s vertex list or NULL if no vertices exist for the object.

**Parameters**

- **object**
  The specified object.
**LObject_Area**

```c
double LObject_Area( LObject pObject );
```

**Description**

Calculates the area of a box, polygon, wire, circle, pie wedge or torus.

**Return Values**

The area of the object in Internal Units squared.

**Parameters**

- `pObject` Specified object.

**See Also**

“LObject_Perimeter” (page 1384), “LObject” (page 1769), “Object Functions” (page 1371)
LObject_Perimeter

\[
double LObject_Perimeter( LObject pObject );
\]

Description

Calculates the perimeter of a box, polygon, wire, circle, pie wedge or torus.

Return Values

The perimeter of the object in Internal Units.

Parameters

\[ pObject \]

Specified object.

See Also

“LObject_Area” (page 1383), “LObject” (page 1769), “Object Functions” (page 1371)
**LObject_GetLayer**

```
LLABEL LObject_GetLayer( LCell pCell, LObject pObject );
```

**Description**

Retrieves the layer of the specified object.

**Return Values**

Returns the layer of the object if successful; otherwise returns NULL.

**Parameters**

- `pCell` Specified cell containing the object.
- `pObject` Specified object.

**See Also**

“LObject” (page 1769), “Object Functions” (page 1371), “LWireParam” (page 1808)


**LObject_GetGDSIIDataType**

```c
short LObject_GetGDSIIDataType( LObject pObject );
```

**Description**

Retrieves the GDSII data type of an object.

**Return Values**

The GDSII data type if successful, -1 if an error occurred such as `pObject` is NULL or `pObject` is an instance.

**Parameters**

- **pObject**
  Specified object.

**Example**

```c
LFile pFile = LFile_GetVisible();
if(Assigned(pFile))
{
    // Get the GDSII Data Type of the first object in the selected.
    LSelection pSelection = LSelection_GetList();
    if(Assigned(pSelection))
    {
        LObject pObject = LSelection_GetObject(pSelection);
        if(Assigned(pObject))
        {
            short iObjectDataType = LObject_GetGDSIIDataType(pObject);
            // More Processing
            // ...
        }
    }
}
```

**Version**

Available in L-Edit 8.2 and later versions.

**See Also**

"LObject_SetGDSIIDataType" (page 1387), "Object Functions" (page 1371), "LObject" (page 1769), "LStatus" (page 1792)
**LObject_SetGDSIIDataType**

```c
LStatus LObject_SetGDSIIDataType( LObject pObject, short GDSIIDataType );
```

**Description**

Sets the GDSII data type of an object.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value with possible values:

-LBadObject — one or more of the following:

- pObject is NULL
- pObject is an instance

**Parameters**

- **pObject** Specified object.
- **GDSIIDataType** GDSII data type

**Example**

```c
LFile pFile = LFile_GetVisible();
if(Assigned(pFile))
{
    LCell pCell = LCell_Find(pFile, "MyCell");
    LLayer pLayer = LLayer_Find(pFile, "Poly");
    if(Assigned(pCell) && Assigned(pLayer))
    {
        LObject pObject;
        // Set the GDSII data type of each object on Poly in cell MyCell to 12.
        for(pObject = LObject_GetList(pCell, pLayer); Assigned(pObject);
            pObject = LObject_GetNext(pObject))
        {
            if(LObject_SetGDSIIDataType(pObject, 12) != LStatusOK)
            {
                // Some problem occurred.
                break;
            }
        }
    }
}
```

**Version**

Available in L-Edit 8.2 and later versions.
See Also

“LObject_GetGDSIIDataType” (page 1386), “Object Functions” (page 1371), “LObject” (page 1769), “LStatus” (page 1792)
**LObject_ChangeLayer**

```c
LStatus LObject_ChangeLayer( LCell pCell, LObject pObject, LLabel pNewLayer );
```

**Description**

Changes the layer of an object to a different layer. After this function is called, the pObject pointer is no longer valid for getting the next object (**LObject_GetNext**—see example below).

**Return Values**

LStatusOK if successful. If an error occurs, **LStatus** contains the error type with possible values:

- **LBadCell**—pCell is NULL.
- **LBadObject**—pObject is NULL, pObject is an instance, or object is corrupted.
- **LBadLayer**—pLayer is NULL.

**Parameters**

- **pCell** Cell which contains the object.
- **pObject** Object to change the layer of.
- **pNewLayer** New layer.

**Example**

```c
LCell pCell = LCell_GetVisible(); // The current cell.
if(NotAssigned(pCell))
{
    LDialog_AlertBox("ERROR: Could not find a Visible Cell.");
    return;
}

LFile pTDBFile = LCell_GetFile(pCell);// The TDB current file.
if(NotAssigned(pTDBFile))
{
    LDialog_AlertBox("ERROR: Could not get the TDB file from the Visible Cell.");
    return;
}

// Change all objects on Poly to Metal1.
LLayer pPoly = LLayer_Find(pTDBFile, "Poly");
LLayer pMetal1 = LLayer_Find(pTDBFile, "Metal1");
if(Assigned(pPoly) && Assigned(pMetal1))
{
    LObject pObject = NULL, pNextObject = NULL;
    for(pObject = LObject_GetList(pCell, pPoly); Assigned(pObject);
    pObject = pNextObject)
    {
```
// After the LObject_ChangeLayer call, the pObject pointer will no longer be valid.
// So get the next object before you change the layer.
pNextObject = LObject_GetNext(pObject);
if(LObject_ChangeLayer(pCell, pObject, pMetal1) == LStatusOK)
    {  // The layer was changed to Metal1.
    }
} // endfor(pObject = LObject_GetList(pCell, pPoly);
   Assigned(pObject); pObject = pNextObject)
}  // endif(Assigned(pLayer))

Version

Available in L-Edit 8.3 and later versions.

See Also

LObject_ConvertToPolygon

LStatus LObject_ConvertToPolygon( LCell pCell, LObject* ArrayOfObjects, unsigned int nNumOfObjects );
LObject_Copy

LObject LObject_Copy( LCell pCell, LLabel pLayer, LObject pObject );
LObject_SnapToGrid

LBoolean LObject_SnapToGrid( LObject pObject, LCoord nGridSize );

Description

LObject_SnapToGrid snaps the object to the grid size specified. This is similar to the command Draw > Convert > Snap to Manufacturing Grid except it snaps to the specified grid rather than the manufacturing grid.

Return Values

LTrue—If the object was not on the specified grid and the object was snapped to the grid (changed).

LFalse—If the object was on the specified grid and the object was not changed.

Parameters

pObject The object to snap.
nGridsize The grid size, in internal units.

Example

LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    LFile pTDBFile= LCell_GetFile(pCell);
    LGrid_v10_00 oGridInfo;
    LFile_GetGrid_v10_00(pTDBFile, &oGridInfo);

    // Snap all selected objects to the mouse snap grid.
    LSelection pSelection = NULL;
    for(pSelection = LSelection_GetList();
        Assigned(pSelection);
        pSelection = LSelection_GetNext(pSelection))
    {
        LObject_ SnapToGrid(LSelection_GetObject(pSelection),
            oGridInfo.mouse_snap_grid_size);
    }
}

Version

Available in L-Edit v13.1 and later versions.
LObject_SnapToMfgGrid

**LBoolean** LObject_SnapToMfgGrid( **LObject** pObject, **LFile** pTDBFile );

**Description**

**LObject_SnapToMfgGrid** snaps the specified object to the manufacturing grid. This is the same as the command **Draw > Convert > Snap to Manufacturing Grid**.

**Return Values**

LTrue—If the object was not on the manufacturing grid and the object was snapped to the manufacturing grid (changed).

LFalse—If the object was on the manufacturing grid and the object was not changed.

**Parameters**

- **pObject**  
  The object to snap.

- **pTDBFile**  
  The TDBFile whose manufacturing grid to snap to.

**Example**

```
LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
   LFile pTDBFile = LCell_GetFile(pCell);
   // Snap all selected objects to the manufacturing grid.
   LSelection pSelection = NULL;
   for(pSelection = LSelection_GetList();
       Assigned(pSelection);
       pSelection = LSelection_GetNext(pSelection))
   {
      LObject_SnapToGrid(LSelection_GetObject(pSelection),
       pTDBFile);
   }
}
```

**Version**

Available in L-Edit v13.1 and later versions.
LObject_DistanceToPoint

\[
\text{LCoord LObject\_DistanceToPoint( LObject pObject, LPoint ptPoint, LFile pTDBFile )};
\]
Vertex Functions

“LVertex_GetCount” (page 1397)  “LVertex_GetArray” (page 1398)
“LObject_GetVertexList” (page 1382)  “LVertex_GetNext” (page 1399)
“LVertex_GetPoint” (page 1400)  “LVertex_SetPoint” (page 1401)
“LVertex_GetCurve” (page 1405)  “LVertex_GetCurveEX” (page 1406)
“LVertex_Add” (page 1402)  “LVertex_Delete” (page 1403)
“LVertex_AddCurve” (page 1404)  “LVertex_RemoveCurve” (page 1410)
“LVertex_HasCurve” (page 1408)  “LVertex_SetCurve” (page 1409)
“LVertex_GetCurveExactCenter” (page 1407)
LVertex_GetCount

long LVertex_GetCount( LObject object );

Description

Gets the number of vertices in a polygon or wire.

Return Values

Returns the number of vertices in object of type polygon or wire; on error returns -1.

Parameters

object

Specified object.

See Also

“LObject” (page 1769), “Object Functions” (page 1371)
LVertex_GetArray

```c
long LVertex_GetArray( LObject object, LPoint point_arr[], const int maxpoints );
```

**Description**

Fills an array with the vertices stored in an object. If the number of vertices is greater than `maxpoints`, the extra vertices are ignored.

**Return Values**

Returns the number of vertices in object of type polygon or wire; on error returns -1.

**Parameters**

- `cell` Specified object.
- `point_arr` Array of vertices.
- `maxpoints` Maximum number of vertices allowed.

**See Also**

“LObject” (page 1769), “LTransform” (page 1801), “Object Functions” (page 1371)
LVertex_GetNext

LVertex LVertex_GetNext( LVertex vertex );

Description

Gets the next vertex of an object.

Return Values

Returns a pointer to the next vertex in a polygon or wire object’s vertex list or NULL if no vertices exist for the object.

Parameters

vertex The previous vertex.

Example

/* for each vertex of the polygon */
for (LVertex Vertex = LObject_GetVertexList (MyPolygon);
    vertex !=NULL;
    Vertex = LVertex_GetNext (Vertex);
{ /* do something with the current vertex */
}
LVertex_GetPoint

LPoint LVertex_GetPoint( LVertex vertex );

Description

Gets the x and y coordinates for a vertex.

Return Values

Returns a point structure containing the coordinates. If the vertex pointer was invalid, the return value is not defined.

Parameters

vertex A specified vertex.
LVertex_SetPoint

```c
LStatus LVertex_SetPoint( LVertex vertex, LPoint point );
```

Description

Sets the x and y coordinates for a vertex.

Return Values

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

Parameters

- **vertex**: A specified vertex.
- **point**: A point structure with the x and y coordinates.

Example

```c
/* get the point information associated with MyVertex */
LPoint Point = LVertex_GetPoint(MyVertex)
/* change the position of the point */
Point.y +=10;
Point.x -=20;

/* update the position of the MyVertex */
LVertex_SetPoint(MyVertex, point);
```
LVertex_Add

LVertex LVertex_Add( LObject object, const LVertex prev_vertex, LPoint point );

Description

Adds a vertex to the object. The object can be an LPolygon or LWire. prev_vertex is a pointer to the previous vertex. If prev_vertex=NULL, the vertex is added to the head of the list.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

object Object to add vertex to.
prev_vertex The previous vertex.
point A point structure with the x and y coordinate.
LVertex_Delete

```c
LStatus LVertex_Delete( LObject object, LVertex vertex );
```

**Description**

Deletes the vertex from the object. The object can be an LPolygon or LWire. This function will delete only if more than three vertices exist.

**Return Values**

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `object` The object to delete the vertex from.
- `vertex` The vertex to be deleted.
LVertex_AddCurve

```c
LStatus LVertex_AddCurve( LObject object, LVertex vertex, LPoint center,
                           LArcDirection dir );
```

**Description**

Adds an arc going from the vertex indicated by the vertex parameter to the next vertex `LVertex_GetNext(vertex)`. The arc is centered at the center parameter and has clockwise direction if `dir` is CW and counterclockwise if `dir` is CCW.

**Return Values**

Returns LStatusOK if successful, LBadObject if object is NULL, or LBadParameters if vertex is either NULL or the last one.

**Parameters**

- **object**: The polygon to add the arc to.
- **vertex**: The start of the arc that ends in `LVertex_GetNext(vertex)`.  
- **center**: The center of the arc.
- **dir**: `CW` for clockwise, `CCW` for counterclockwise.

**Example**

```c
LVertex_AddCurve(pPolygon, pStartVertex, Center, CCW);
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

**LVertex_GetCurve**

```c
LStatus LVertex_GetCurve( LVertex vertex, LPoint *center, LArcDirection *dir );
```

**Description**

Retrieves the center and the direction of the curve that starts from the indicated vertex.

**Return Values**

Returns LStatusOK if successful, LBadParameters upon failure.

**Parameters**

- `vertex` The start vertex of the curve.
- `center` The destination for the center.
- `dir` The destination for the direction, either `CW` for clockwise or `CCW` for counterclockwise.

**Example**

```c
LPoint Center;
LArcDirection Dir;
LVertex_GetCurve(pVertex, &Center, &Dir);
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LVertex_GetCurveEX” (page 1406), “LVertex_SetCurve” (page 1409), “LVertex_AddCurve” (page 1404)
LVertex_GetCurveEX

LStatus LVertex_GetCurveEX( LObject object, LVertex vertex, LPoint *center,
      LCoord *radius, LPoint *start, LPoint *end, LArcDirection *dir );

Description

Retrieves the parameters of the curve that starts from indicated vertex.

Return Values

Returns LStatusOK if successful, LBadParameters on failure.

Parameters

- **object**: The curved polygon.
- **vertex**: The start vertex of the curve.
- **center**: The destination for the center.
- **radius**: The destination for the radius.
- **start**: The destination for the start of the arc.
- **end**: The destination for the end of the arc.
- **dir**: The destination for the direction, either CW for clockwise or CCW for counterclockwise.

Example

LPoint Center, Start, End;
LCoord Radius;
LArcDirection Dir;
LVertex_GetCurve(pPolygon, pVertex, &Center, &Radius, &Start, &End, &Dir);

Version

Available in L-Edit 8.4 and later versions.

See Also

“LVertex_GetCurve” (page 1405), “LVertex_SetCurve” (page 1409), “LVertex_AddCurve” (page 1404)
**LVertex_GetCurveExactCenter**

```c
DPoint LVertex_GetCurveExactCenter( LObject object, LVertex vertex, LCoord radius, LArcDirection *dir );
```

**Description**

Retrieves the center and the direction of the curve that starts from the indicated vertex.

**Return Values**

Exact center as DPoint. Unlike LPoint, DPoint has coordinates as doubles, not integers.

**Parameters**

- **object**
  - The curved polygon.
- **vertex**
  - The start vertex of the curve.
- **radius**
  - The desired radius.
- **dir**
  - (This is supposed to be the destination for the direction but as of L-Edit version 8.4 is not in use.)

**Example**

```c
LArcDirection Dir = CCW;
LPoint Center;
DPoint ExactCenter = LVertex_GetCurveExactCenter(pPolygon, pVertex, Radius, &Dir);
Center.x = ExactCenter.x;
Center.x = ExactCenter.y;
LVertex_AddCurve(pPolygon, pVertex, Center, Dir);
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LVertex_GetCurve” (page 1405), “LVertex_SetCurve” (page 1409), “LVertex_AddCurve” (page 1404)
LVertex_HasCurve

```c
int LVertex_HasCurve( LVertex vertex );
```

**Description**

Indicates whether the edge that follows the vertex indicated in the parameter `vertex` is a curve.

**Return Values**

1 if the edge following the vertex is a curve, 0 otherwise.

**Parameters**

- `vertex`: The start vertex of the edge to check.

**Example**

```c
int bIsCurve = LVertex_HasCurve(pVertex);
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LPolygon_HasCurve” (page 1449)
**LVertex_SetCurve**

```c
LStatus LVertex_SetCurve( LVertex vertex, LPoint center, LArcDirection dir );
```

**Description**

Sets the center and the direction of the curve that starts from the indicated vertex.

**Return Values**

Returns LStatusOK if successful, LBadParameters on failure.

**Parameters**

- **vertex**
  The start vertex of the curve.
- **center**
  The center of the curve.
- **dir**
  The direction, either *CW* for clockwise or *CCW* for counterclockwise.

**Example**

```c
LPoint Center;
LArcDirection Dir;
Center.x = 50;
Center.y = 100;
Dir = CCW ;
LVertex_SetCurve(pVertex, Center, Dir);
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LVertex_GetCurve” (page 1405), “LVertex_GetCurveEX” (page 1406), “LVertex_AddCurve” (page 1404),
“LVertex_GetCurveExactCenter” (page 1407)
LVertex RemoveCurve

LStatus LVertex_RemoveCurve(LObject object, LVertex vertex);

Description

Straightens the edge that follows the vertex indicated in the parameter.

Return Values

Returns LStatusOK if successful, LBadParameters if the vertex is either NULL or the last one.

Parameters

object The curved polygon.
vertex The start vertex of the edge to straighten.

Example

LVertex_RemoveCurve(pPolygon, pVertex);

Version

Available in L-Edit 8.4 and later versions.

See Also

“LVertex_HasCurve” (page 1408), “LVertex_AddCurve” (page 1404),
“LVertex_RemoveCurve” (page 1410), “LPolygon_RemoveAllCurves” (page 1450)
Box Functions

“LBox_New” (page 1412)
“LBox_Set” (page 1413)
“LBox_GetRect” (page 1414)
LBox_New

```c
LObject LBox_New( LCell cell, LLabel layer,
                    LCoord x0, LCoord y0, LCoord x1, LCoord y1 );
```

**Description**

Creates a new box object in `cell` on `layer` with the given coordinates.

**Return Values**

Returns a pointer to the newly created box if successful; otherwise returns NULL.

**Parameters**

- `cell` Cell where box will be drawn.
- `layer` Layer on which the box will be drawn.
- `x0` Lower left x-coordinate of box.
- `y0` Lower left y-coordinate of box.
- `x1` Upper right x-coordinate of box.
- `y1` Upper right y-coordinate of box.

**See Also**

“LObject” (page 1769), “LWireParam” (page 1808), “Box Functions” (page 1411)
**LBox_Set**

```c
LStatus LBox_Set( LCell cell, LObject object, LRect box );
```

**Description**

Modifies the coordinates of the object in `cell` according to the specification contained in `box`.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, `LStatus` contains the error value.

**Parameters**

- `cell` Cell that contains the box.
- `object` Pointer to the box object.
- `box` New coordinates of the box.

**See Also**

“`LStatus`” (page 1792), “`LObject`” (page 1769), “`LTransform`” (page 1801), “`Box Functions`” (page 1411)
**LBox_GetRect**

\[
\text{LRect LBox_GetRect( LObject object );}
\]

**Description**

Returns the minimum bounding box (MBB) of the specified box.

**Return Values**

If successful, an **LRect** structure containing the minimum bounding box (MBB) of the specified box; on error, a rectangle whose coordinates are all zeros.

**Parameters**

- **object**: Specified box object.

**See Also**

“**LObject**” (page 1769), “**LTransform**” (page 1801), “**Box Functions**” (page 1411)
Circle Functions

“LCircle_New” (page 1416)  “LCircle_Set” (page 1417)
“LCircle_GetRadius” (page 1419)  “LCircle_GetCenter” (page 1418)
“LCircle_GetRect” (page 1420)
LCircle_New

\small\texttt{LObject LCircle\_New( LCell cell, LLabel layer, LPoint center, LCoord radius );}\normalsize

Description

Creates a new circle in \textit{cell} on \textit{layer} with the center and radius specified by \textit{center} and \textit{radius}.

Return Values

Returns a pointer to the newly created circle if successful; otherwise returns NULL.

Parameters

- \textit{cell} \hspace{1cm} Cell where the new circle is to be drawn.
- \textit{layer} \hspace{1cm} Layer on which circle is to be drawn.
- \textit{center} \hspace{1cm} x- and y- coordinates of the center.
- \textit{radius} \hspace{1cm} Radius of the circle.

See Also

"LObject" (page 1769), "LWireParam" (page 1808), "LTransform" (page 1801), "Circle Functions" (page 1415)
LCircle_Set

\[
\text{LStatus } \text{LCircle_Set}( \text{LCell cell, LObject object, LPoint center, LCoord radius });
\]

**Description**

Modifies object in `cell` to the new `center` and `radius`.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- **cell**
  Cell where the circle is drawn.
- **object**
  Circle object.
- **center**
  New x- and y- coordinates of the center.
- **radius**
  New circle radius.

**See Also**

LCircle_GetCenter

LPoint LCircle_GetCenter( LObject object );

Description

Gets the coordinates of the center of a circle.

Return Values

Returns the center point of object, or (0,0) on error

Parameters

object Circle object.

See Also

“LTransform” (page 1801), “LObject” (page 1769), “Circle Functions” (page 1415)
LCircle_GetRadius

LCoord LCircle_GetRadius( LObject pObject );

Description

Gets the radius of a circle.

Return Values

Returns the radius of object, or (0,0) on error

Parameters

*pObject Circle object.

See Also

“LCoord” (page 1707), “LObject” (page 1769), “Circle Functions” (page 1415)
LCircle_GetRect

```c
LRect LCircle_GetRect( LObject pObject );
```

Description

Returns the minimum bounding box (MBB) of the specified circle.

Return Values

If successful, an `LRect` structure containing the minimum bounding box (MBB) of the specified circle; on error, a rectangle whose coordinates are all zeros.

Parameters

- `pObject` Specified circle object.

See Also

“`LRect`” (page 1783), “`LObject`” (page 1769), “Circle Functions” (page 1415)
Torus and Pie Functions

“LPie_CreateNew” (page 1422)

“LPie_GetParams” (page 1424)    “LPie_SetParams” (page 1425)

“LTorus_CreateNew” (page 1427)

“LTorus_GetParams” (page 1429)    “LTorus_SetParams” (page 1430)

“LTorusParams” (page 1800)
LPie_CreateNew

LObject LPie_CreateNew( LCell cell, LLabel layer, LPieParams *params );

Description

Creates a new pie in cell on layer with the parameters specified by params.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadCell</td>
<td>cell is NULL</td>
</tr>
<tr>
<td>LBadParameters</td>
<td>One or more of the following errors:</td>
</tr>
<tr>
<td></td>
<td>- layer is NULL</td>
</tr>
<tr>
<td></td>
<td>- params is NULL</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>- params.Radius ≤ 0 or params.Radius &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>- params.StartAngle &lt; 0 or params.StartAngle &gt; 360</td>
</tr>
<tr>
<td></td>
<td>- params.StopAngle &lt; 0 or params.StopAngle &gt; 360</td>
</tr>
</tbody>
</table>

Parameters

- cell: Cell where the new pie is to be drawn.
- layer: Layer on which the new pie is to be drawn.
- params: Parameters of the pie.

Example

/* Define parameters of the new pie */
LPieParams pParams;
pParams.Center = {0, 0};
pParams.Radius = 50;
pParams.StartAngle = 45;
pParams.StopAngle = 90;

/* Use these parameters to define a new pie */
LObject MyPie = LPie_CreateNew(pCell, pLayer, &pParams);

/* Duplicate MyPie in a new cell and layer */
LPie_GetParams(MyPie, &pParams);
LObject NewPie = LPie_CreateNew(newCell, newLayer, &pParams);
Version

Available in L-Edit 8.4 and later versions.

See Also

“LPie_GetParams” (page 1424), “LPie_SetParams” (page 1425), “LPieParams” (page 1779)
LPie_GetParams

```c
LStatus LPie_GetParams( LObject object, LPieParams *params );
```

**Description**

Retrieves the parameters of the specified pie and writes them to the destination `params`.

**Return Values**

Returns `LStatusOK` if successful. If an error occurs, `LStatus` contains the error type with `LBadParameters`. An error occurs when either `object` or `params` is NULL.

**Parameters**

- `object` Specified pie object.
- `params` Destination to which pie parameters are written.

**Example**

```c
/*Write parameter values of MyPie to destination pParams*/
LPieParams pParams;
LPie_GetParams(MyPie, &pParams);
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LPie_SetParams” (page 1425), “LPie_CreateNew” (page 1422), “LPieParams” (page 1779)
LPie_SetParams

```c
LStatus LPie_SetParams( LCell cell, LObject object, LPieParams *params );
```

**Description**

Sets the parameters of the specified pie on `cell` to the values defined by `params`.

**Return Values**

Returns `LStatusOK` if successful. If an error occurs, `LStatus` contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadCell</td>
<td><code>cell</code> is NULL</td>
</tr>
<tr>
<td>LBadParameters</td>
<td>One or more of the following errors:</td>
</tr>
<tr>
<td></td>
<td>- <code>object</code> is NULL</td>
</tr>
<tr>
<td></td>
<td>- <code>params</code> is NULL</td>
</tr>
<tr>
<td></td>
<td>- `</td>
</tr>
<tr>
<td></td>
<td>- <code>params.Radius ≤ 0</code> or <code>params.Radius &gt; WORLD_MAX</code></td>
</tr>
<tr>
<td></td>
<td>- <code>params.StartAngle &lt; 0</code> or <code>params.StartAngle &gt; 360</code></td>
</tr>
<tr>
<td></td>
<td>- <code>params.StopAngle &lt; 0</code> or <code>params.StopAngle &gt; 360</code></td>
</tr>
</tbody>
</table>

**Parameters**

- `cell`  
  Cell containing the pie.
- `object`  
  Specified pie object.
- `params`  
  Pie parameter values.

**Example**

```c
/* Get the current parameters of MyPie */
LPieParams pParams;
LPie_GetParams(MyPie, &pParams);

/* Change the radius of the pie */
pParams.Radius *= 2;

/* Update MyPie with the new parameters */
LPie_SetParams(pCell, MyPie, &pParams)
```
Version

Available in L-Edit 8.4 and later versions.

See Also

“LPie_GetParams” (page 1424), “LPie_CreateNew” (page 1422), “LPieParams” (page 1779)
LTorus_CreateNew

**Description**

Creates a new torus in cell on layer with the parameters specified by params.

**Return Values**

Returns LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadCell</td>
<td>cell is NULL</td>
</tr>
<tr>
<td>LBadParameters</td>
<td>One or more of the following errors:</td>
</tr>
<tr>
<td></td>
<td>▪ layer is NULL</td>
</tr>
<tr>
<td></td>
<td>▪ params is NULL</td>
</tr>
<tr>
<td></td>
<td>▪</td>
</tr>
<tr>
<td></td>
<td>▪ params.Center ( LCoord x, LCoord y )</td>
</tr>
<tr>
<td></td>
<td>▪ params.InnerRadius ≤ 0 or params.InnerRadius &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>▪ params.OuterRadius ≤ 0 or params.OuterRadius &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>▪ params.StartAngle &lt; 0 or params.StartAngle &gt; 360</td>
</tr>
<tr>
<td></td>
<td>▪ params.StopAngle &lt; 0 or params.StopAngle &gt; 360</td>
</tr>
</tbody>
</table>

**Parameters**

- **cell**
  Cell where the new torus is to be drawn.

- **layer**
  Layer on which the new torus is to be drawn.

- **params**
  Torus parameter values.

**Example**

```c
/* Define parameters for a torus */
LTorusParams tParams;
tParams.ptCenter = LPoint_Set (0,0);
tParams.nInnerRadius = 50;
tParams.nOuterRadius = 100;
tParams.dStartAngle = 45;
tParams.dStopAngle = 90;

/* Use these parameters to define a new torus */
```
LObject MyTorus = LTorus_CreateNew(tCell, tLayer, &tParams);

/* Duplicate MyTorus in a new cell and layer */
LTorus_GetParams(MyTorus, &tParams);
LObject NewTorus = LTorus_CreateNew(newCell, newLayer, &tParams);

Version

Available in L-Edit 8.4 and later versions.

See Also

LTorus_GetParams

\begin{verbatim}
LStatus LTorus_GetParams(LObject object, LTorusParams *params);
\end{verbatim}

Description

Retrieves the parameters of the specified torus and writes them to the destination \texttt{params}.

Return Values

Returns \texttt{LStatusOK} if successful. If an error occurs, returns the error value \texttt{LBadParameters}. An error occurs when \texttt{object} or \texttt{params} is NULL.

Parameters

\begin{itemize}
\item \texttt{object} Specified torus object.
\item \texttt{params} Destination to which torus parameters are written.
\end{itemize}

Example

\begin{verbatim}
/*Write parameter values of MyTorus to destination tParams*/
LtorusParams tParams;
LTorus_GetParams(MyTorus, &tParams);
\end{verbatim}

Version

Available in L-Edit 8.4 and later versions.

See Also

“LTorus_SetParams” (page 1430), “LTorus_CreateNew” (page 1427), “LTorusParams” (page 1800)
LTorus_SetParams

LStatus LTorus_SetParams(LCell cell, LObject object, LTorusParams *params);

Description

Sets the parameters of the specified torus on cell to the values defined in params.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadCell</td>
<td>cell is NULL</td>
</tr>
<tr>
<td>LBadParameters</td>
<td>One or more of the following errors:</td>
</tr>
<tr>
<td></td>
<td>• object is NULL</td>
</tr>
<tr>
<td></td>
<td>• params is NULL</td>
</tr>
<tr>
<td></td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>• params.InnerRadius ≤ 0 or params.InnerRadius &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>• params.OuterRadius ≤ 0 or params.OuterRadius &gt; WORLD_MAX</td>
</tr>
<tr>
<td></td>
<td>• params.StartAngle &lt; 0 or params.StartAngle &gt; 360</td>
</tr>
<tr>
<td></td>
<td>• params.StopAngle &lt; 0 or params.StopAngle &gt; 360</td>
</tr>
</tbody>
</table>

Parameters

- **cell** Cell containing the torus.
- **object** Specified torus object.
- **params** Torus parameter values.

Example

/* Get the current parameters of MyTorus */
LTorusParams tParams;
LTorus_GetParams(MyTorus, &tParams);

/* Double the outer radius of the torus */
tParams.OuterRadius *= 2;

/* Update MyTorus with the new parameters */
LTorus_SetParams(tCell, MyTorus, &tParams)
Version

Available in L-Edit 8.4 and later versions.

See Also

Wire Functions

“LWire_New” (page 1433)
“LWireGetWidth” (page 1434)
“LWireGetCapType” (page 1435)
“LWireGetJoinType” (page 1436)
“LWireGetMiterAngle” (page 1437)
“LWireGetLength” (page 1438)
“LWireGetSquares” (page 1439)
“LWireGetResistance” (page 1440)
“LWireSetWidth” (page 1441)
“LWireSetCapType” (page 1443)
“LWireSetJoinType” (page 1442)
“LWireSetMiterAngle” (page 1444)
LWire_New

LObject LWire_New( LCell cell, LLabel layer, LWireConfig *config, LWireConfigBits bits, LPoint point_arr[], const int npoints );

Description

Creates a new wire in cell on layer. The new wire will have npoints set to the values in the array point_arr. If config is NULL or bits is zero, the wire will have the default width, join, and end styles of the corresponding layer. If bits is set to a mask of LWireConfigBits enum values, then values from the structure config will be used to override the defaults for the settings of bits.

Return Values

Pointer to the newly created wire if successful; NULL otherwise.

Parameters

- **cell**: Cell which will contain the wire.
- **layer**: Wire layer.
- **config**: Pointer to the wire configuration structure.
- **bit**: Wire configuration bits.
- **point_arr**: Array of wire vertices.
- **npoints**: Number of wire vertices.

See Also

LWireGetWidth

LCoord LWireGetWidth( LObject object );

Description

Gets the wire width.

Return Values

Returns the width setting of the object, or zero on error.

Parameters

object Specified wire object.

See Also

“LObject” (page 1769), “LTransform” (page 1801), “Wire Functions” (page 1432)
LWire_GetCapType

\texttt{LCapType} \textbf{LWire\_GetCapType} ( \texttt{LObject} \ pObject \ ) ;

\textbf{Description}

Gets the wire cap type.

\textbf{Return Values}

Returns the wire cap style of object. The return value is undefined on error.

\textbf{Parameters}

\texttt{pObject} \quad \text{Wire object.}

\textbf{See Also}

“\texttt{LCapType}” (page 1699), “\texttt{LObject}” (page 1769), “\texttt{Wire Functions}” (page 1432)
LWire_GetJoinType

LJoinType LWire_GetJoinType( LObject object );

Description
Gets the wire join type

Return Values
Returns the wire join style of object—miter, round, or bevel. The return value is undefined on error.

Parameters

object Wire object.

See Also
“LPort” (page 1781), “LObject” (page 1769), “Wire Functions” (page 1432)
LWire_GetMiterAngle

```c
short LWire_GetMiterAngle( LObject Object );
```

**Description**

Gets the wire miter angle

**Return Values**

Returns the miter angle of object. It returns -1 on error.

**Parameters**

- `object` Wire object.

**See Also**

- **LObject (page 4-698)**, **Wire Functions (page 4-400)**
LWire_GetLength

```c
double LWire_GetLength( LObject pObject )
```

**Description**

Calculates the centerline length of the wire including end styles.

**Return Values**

The centerline length of the wire in Internal Units.

**Parameters**

- `pObject` Specified object.

**See Also**

LWire_GetSquares

    double LWire_GetSquares( LObject pObject );

Description

Calculates the number of squares of an orthogonal wire including end styles. In the calculation of the number of squares, corners are counted as \( \frac{1}{2} \) a square.

Return Values

The number of squares of an orthogonal wire in Internal Units. If the object is not an orthogonal wire, then zero.

Parameters

- **pObject** Specified object.

See Also

LWire_GetResistance

double LWire_GetResistance( LObject pObject );

Description

Calculates the resistance of an orthogonal wire including end styles. This uses the Resistivity on the Setup Layers dialog and the number of squares of the wire. In the calculation of the number of squares, corners are counted as ½ a square.

Return Values

The resistance of an orthogonal wire in Ohms. If the object is not an orthogonal wire, then zero.

Parameters

pObject Specified object.

See Also

LWire_SetWidth

LStatus LWire_SetWidth( LCell pCell, LObject pObject, LCoord nWidth );
LWire_SetJoinType

LStatus LWire_SetJoinType( LCell pCell, LObject pObject, LJoinType eJoinType );
LWire_SetCapType

LStatus LWire_SetCapType( LCell pCell, LObject pObject, LCapType eCapType );
LWire_SetMiterAngle

LStatus LWire_SetMiterAngle(LCell pCell, LObject pObject, short nAngle);
Polygon Functions

“LPolygon_New” (page 1446)
“LPolygon_HasCurve” (page 1449)
“LPolygon_RemoveAllCurves” (page 1450)  “LPolygon_StraightenAllCurves” (page 1451)

Obsolete:
“LPolygon_WireToPolygon” (page 1447)
“LPolygon_CircleToPolygon” (page 1448)
LPolygon_New

\[ LObject \ LPolygon\_New( \ LCell \ cell, \ LLabel \ layer, \ LPoint \ point\_arr[], \ const \ int \ npoints \ ); \]

**Description**

Creates a new polygon object in `cell` on `layer`. The new polygon will have `npoints` vertices at locations specified in `point_arr`.

**Return Values**

Returns a pointer to the newly created polygon if successful; NULL otherwise.

**Parameters**

- `cell` Cell which will contain the polygon.
- `layer` Wire layer.
- `point_arr` Array of polygon vertices.
- `npoints` Number of polygon vertices.

**See Also**

LPolygon_WireToPolygon

```c
LObject LPolygon_WireToPolygon( LCell cell, LLabel layer, LObject object );
```

**Description**

Converts a wire object to a polygon object.

**Return Values**

Returns a pointer to the newly converted polygon if successful; NULL otherwise.

**Parameters**

- `cell` Cell containing the wire object.
- `layer` Wire layer.
- `object` Wire object.

**Note:** Note that this function is superseded in L-Edit V10 and later.

**See Also**

“LObject” (page 1769), “LWireParam” (page 1808), “Polygon Functions” (page 1445)
**LPolygon_CircleToPolygon**

```c
LObject LPolygon_CircleToPolygon( LCell cell, LLabel layer, 
        LObject object, int NumSides );
```

**Description**

Converts a circle to a polygon with the given number of sides.

**Return Values**

Returns a pointer to the newly converted polygon if successful; NULL otherwise.

**Parameters**

- **cell** Cell containing the circle.
- **layer** Circle layer.
- **object** Circle object.
- **NumSides** Number of sides in the new polygon.

**Note:**

Note that this function is superseded in L-Edit V10 and later.

**See Also**

“LObject” (page 1769), “LWireParam” (page 1808), “Polygon Functions” (page 1445)
**LPolygon_HasCurve**

```c
int LPolygon_HasCurve( LObject object );
```

**Description**

Indicates whether the specified polygon contains curves.

**Return Values**

1 if the polygon does contain curves; 0 otherwise.

**Parameters**

- `object` The polygon to be checked for curves.

**Example**

```c
int flagCurves = LPolygon_HasCurve(MyPolygon);
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LPolygon_RemoveAllCurves” (page 1450), “LPolygon_StraightenAllCurves” (page 1451)
**LPolygon_RemoveAllCurves**

```c
LStatus LPolygon_RemoveAllCurves( LObject object );
```

**Description**

Removes all curved edges from the specified polygon and replaces them with straight edges. For a better linear approximation of the curved edges, use “**LPolygon_StraightenAllCurves**” (page 1451).

**Return Values**

Returns `LStatusOK` if successful. If an error occurs, returns the error value `LBadParameters`.

**Parameters**

- `object`  
  Polygon from which all curves are removed.

**Example**

```
LPolygon_RemoveAllCurves(MyPolygon)
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“**LPolygon_HasCurve**” (page 1449), “**LPolygon_StraightenAllCurves**” (page 1451)
LPolygon_StraightenAllCurves

```c
LStatus LPolygon_StraightenAllCurves( LCell cell, LObject object );
```

**Description**

Removes all curved edges from the specified polygon on `cell`, and replaces them with an approximation consisting of straight segments.

**Return Values**

Returns `LStatusOK` if successful. If an error occurs, returns the error value `LBadParameters`.

**Parameters**

- `cell` Cell containing the specified polygon.
- `object` Polygon in which to straighten curves.

**Example**

```c
/* Get the current curve properties of the file */
LCurve CurveSetup;
LFile_GetCurveSetup(MyFile, &CurveSetup);

/* Change the curve properties as desired */
CurveSetup.max_segment_per_curve = 100;
CurveSetup.max_length_of_segment = 50;

/* Assign the properties defined in CurveSetup to MyFile */
LFile_SetCurveSetup(pFile, &CurveSetup);

/* Replace curves in MyPolygon with straight segments*/
LPolygon_StraightenAllCurves(pCell, MyPolygon);
```

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

- “LPolygon_HasCurve” (page 1449), “LPolygon_RemoveAllCurves” (page 1450),
- “Cell Functions” (page 1260)
Port Functions

“LPort_New” (page 1453)
“LPort_Find” (page 1455)
“LPort_GetList” (page 1457)
“LPort_GetText” (page 1459)
“LPort_GetTextSize” (page 1461)
“LPort_GetTextAlignment” (page 1468)
“LPort_GetRect” (page 1464)
“LPort_GetLayer” (page 1462)
“LPort_GetDefaultTextAlignment” (page 1472)

“LLable_New” (page 1474)
“LLable_SetName” (page 1476)
“LPort_GetObjectsList” (page 1478)
“LPort_AddObject” (page 1480)
“LObject_GetPort” (page 1482)
“LPort_DeletePortOnly” (page 1484)
“LPort_SetAccessDirection” (page 1487)
“LPort_GetMustConnect” (page 1489)
“LPort_GetPinName” (page 1491)
“LLayer_FindPurposeName” (page 1493)
LPort_New

```c
LPort LPort_New( LCell cell, LLabel layer, char* text, LCoord x0, LCoord y0,
                 LCoord x1, LCoord y1 );
```

**Description**

Creates a new legacy port in `cell` on legacy `layer` with the specified text and rectangle (location) coordinates `x0, y0, x1, y1`.

On non-legacy layers, creates a new port, new box and new label with specified text. Box coordinates will be `x0, y0, x1, y1`. The label's initial location and alignment is done the same way as for legacy ports.

This function also links the box and the label to the port.

**Return Values**

Pointer to the newly created port if successful; NULL otherwise.

**Parameters**

- `cell` Cell that will contain the port.
- `layer` Port layer.
- `text` Port text string.
- `x0` Lower left x-coordinate of port rectangle.
- `y0` Lower left y-coordinate of port rectangle.
- `x1` Upper right x-coordinate of port rectangle.
- `y1` Upper right y-coordinate of port rectangle.

**Version Compatibility**

Updated in version 16.00 for new port and layer-purpose pair features.

**See Also**

LPort_Delete

LStatus LPort_Delete( LCell cell, LPort port );

Description

Deletes the specified port and all objects that are linked to it from the given cell.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

- cell    Cell containing the port.
- port    Port to be deleted.

Version

Updated in version 16.00 for new port behavior.

See Also

LPort_Find

```c
LPort LPort_Find( LCell cell, const char* name );
```

**Description**

Finds the first port in `cell` with the name specified in `name`.

**Return Values**

Pointer to the port if successful; NULL otherwise.

**Parameters**

- `cell`  
  Cell containing the port.
- `name`  
  Port string to search for.

**Version**

This UPI function was updated in version 16:00 to work with both Electrical and legacy ports.

**See Also**

- “LPort” (page 1781), “LObject” (page 1769), “Port Functions” (page 1452)
LPort_FindNext

```c
LPort LPort_FindNext( LPort pPort, const char* szName );
```

Description

Finds the next port after `pPort` that has the name specified in `szName`.

Return Values

Pointer to the port if successful; NULL otherwise.

Parameters

- `pPort` Specified port.
- `szName` Port string to search for.

Version

This UPI function was updated in version 16:00 to work with both Electrical and legacy ports.

See Also

“LPort” (page 1781), “Port Functions” (page 1452)
LPort_GetList

LPort LPort_GetList( LCell cell );

Description

Gets a pointer to the first port in the given cell.

Return Values

Pointer to the head of the port list if successful; NULL otherwise.

Parameters

cell  Specified cell.

Version

This UPI function was updated in version 16:00 to work with both Electrical and legacy ports.

See Also

“LPort” (page 1781), “LObject” (page 1769), “Port Functions” (page 1452)
LPort_GetNext

LPort LPort_GetNext( LPort port );

Description

Gets a pointer to the port immediately following port in the port list.

Return Values

Pointer to the next element in the port list if successful; NULL otherwise.

Parameters

port Specified port.

Version

This UPI function was updated in version 16:00 to work with both Electrical and legacy ports.

See Also

“LPort” (page 1781), “Port Functions” (page 1452)
LPort_GetText

    char* LPort_GetText( LPort port, char* name, const int maxlen );

Description

    Gets the text of a port. It the port text is longer than maxlen, the extra characters are ignored.

Return Values

    Pointer to the port text buffer if successful; NULL otherwise.

Parameters

    port        Port whose text is required.
    name        String (buffer) containing the port text.
    maxlen      Maximum length allowed for port text.

Version

    This UPI function was updated in version 16:00 to work with both Electrical and legacy ports.

See Also

    “LPort” (page 1781), “Port Functions” (page 1452)
LPort_SetText

```c
char* LPort_SetText( LCell cell, LPort port, char* text, LCoord textsize );
```

**Description**

Sets the text of a port.

**Return Values**

Pointer to the port text string if successful; NULL otherwise.

**Parameters**

- `cell` Cell containing the port.
- `port` Port whose text is being modified.
- `text` String (buffer) containing the port text.
- `textsize` Port text size.

**Version**

This UPI function was updated in version 16:00 to work with both Electrical and legacy ports.

**See Also**

“LPort” (page 1781), “LObject” (page 1769), “LTransform” (page 1801), “Port Functions” (page 1452)
LPort_GetTextSize

LCoord LPort_GetTextSize( LPort port );

Description

Gets the port text size.

Return Values

The port text size if successful; zero on error

Parameters

port Specified port.

Version

Note: This function should be used only for legacy ports.

If used with non-legacy ports LPort_GetTextSize will perform no action, and will return NULL.

See Also

LPort_GetLayer

**LLabel** LPort_GetLayer( **LPort** port );

**Description**

Gets the layer that a port is drawn on.

For Electrical ports LPort_GetLayer will return a pointer to **LLayer** if all the objects (excluding Labels) are on the same layer; otherwise returns NULL.

**Return Values**

Pointer to the port layer if successful; NULL otherwise.

**Parameters**

*port*  
Specified port.

**Version**

---

**Note:** This function was enhanced in version 16.00 to support Electrical ports. Please note new behavior when the input argument is an Electrical port.

---

**See Also**

“**LWireParam**” (page 1808), “**LPort**” (page 1781), “**Port Functions**” (page 1452)
**LPort_GetMbb**

```
LRect LPort_GetMbb( LPort port );
```

**Description**

Gets the minimum bounding box (Mbb) of a port.

**Return Values**

The minimum bounding box if successful, or on error a rectangle whose coordinates are all zeros.

**Parameters**

- `port` Specified port.

**Version**

This UPI function was updated in version 16:00 to work with both Electrical and legacy ports.

**See Also**

“LTransform” (page 1801), “LPort” (page 1781), “Port Functions” (page 1452)
LPort_GetRect

LRect LPort_GetRect( LPort port );

Description

For legacy ports, returns the rectangle (location) of the port.

For non-legacy ports, returns the MBB of all objects excluding text labels.

If only text label(s) are linked to the port than will return MBB of the anchor points of linked label(s).

Return Values

If successful, an LRect structure containing the location of the port; on error, a rectangle whose coordinates are all zeros.

Parameters

port Specified port.

Version

Updated in version 16.00 for new port behavior.

See Also

“LTransform” (page 1801), “LPort” (page 1781), “Port Functions” (page 1452)
LPort_Set

LStatus LPort_Set( LCell cell, LPort port, LRect rect );

Description

Modifies the rectangle (location) of the specified port in the specified cell according to the value specified in rect.

For Electrical ports, if the port contains exactly one box and one or no label LPort_Set will set the size of that box to the given rectangle. If not, LPort_Set returns LBadParameters.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

cell Cell containing the port.
port Port to be modified.
rect New location of the port.

Version

Note: This function was enhanced in version 16.00 to support Electrical ports. Please note new behavior when the input argument is an Electrical port.

See Also

LPort_SetTextSize

LStatus LPort_SetTextSize( LPort pPort, LCoord lcTextSize );

Description

Sets the text size of a port.

Note: This function has been enhanced to support Electrical ports. For Electrical ports, LPort_SetTextSize will set the size of all available labels in the port. If the port does not contain any labels, the return value will be LBadParameters.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value LBadParameters—pPort is NULL.

For Electrical ports, if the port does not contain any labels, the return value will be LBadParameters.

Parameters

pPort Specified port.

lcTextSize Text size in Internal Units.

Example

LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    LPort pPort = LPort_Find(pCell, "Gnd");
    if(Assigned(pPort))
    {
        if(LPort_SetTextSize(pPort, LFile_LocUtoIntU(LCell_GetFile(pCell), 2.5)) == LStatusOK)
        {
            // More Processing
            // ...
        } // endif(LPort_SetTextSize(pPort, LFile_LocUtoIntU(LCell_GetFile(pCell), 2.5)) == LStatusOK)
    } // endif(Assigned(pPort))
} // endif(Assigned(pCell))

Version

Available in L-Edit 8.2 and later versions.

Note: This function was enhanced in version 16.00 to support Electrical ports. Please note new behavior if the input argument is an Electrical port.
See Also

LPort_GetTextAlignment

```c
int LPort_GetTextAlignment( LPort pPort );
```

Description

This function returns port text alignment.

**Note:**
This function has been enhanced to support Electrical ports.

Return Values

Returns an OR of port text alignment attributes if successful, -1 in case of failure. See “Port Alignment Definitions” on page 1471 for details.

For Electrical ports, LPort_GetTextAlignment returns the text alignment if the label is in one of 9 available positions (combination of left/middle/right and bottom/center/top) and has appropriate alignment. Returns -1 otherwise, or if there are no labels in the port.

Parameters

- `pPort` Specified port.

Examples

Use the following for precise port text alignment:

```c
int nAlignment = LPort_GetTextAlignment(pPort);
switch(nAlignment) { ... }
```

If you wish to find out whether the port text is on the bottom (regardless of the horizontal placement or orientation) do this:

```c
int nAlignment = LPort_GetTextAlignment(pPort);
int nVerticalPlacement = nAlignment &
    (PORT_TEXT_TOP|PORT_TEXT_CENTER|PORT_TEXT_BOTTOM);
if(nVerticalPlacement == PORT_TEXT_BOTTOM) { ... }
```

Version

Available in L-Edit 8.3 and later versions.

**Note:**
This function was enhanced in version 16.00 to support Electrical ports. Please note new behavior if the input argument is an Electrical port.
See Also

“LPort_SetTextAlignment” (page 1470), “Port Alignment Definitions” on page 1471.
**LPort_SetTextAlignment**

```c
LStatus LPort_SetTextAlignment( LPort pPort, int nAlignment );
```

**Description**

This function sets port text alignment.

**Note:**

This function has been enhanced to support Electrical ports. For Electrical ports, `LPort_SetTextAlignment` will set the text alignment of all available labels in the port.

**Return Values**

Returns `LStatusOK` if successful. If an error occurs, returns the error value `LBadParameters`.

For Electrical ports, if the port doesn't contain any labels, the return value will be `LBadParameters`.

**Parameters**

- **pPort**: Specified port.
- **nAlignment**: An OR of port text alignment attributes (see “Port Alignment Definitions” on page 1471 for details).

**Example**

If you want to place the text of `pPort` on the right bottom of the box use the following:

```c
LPort_SetTextAlignment(pPort, PORT_TEXT_RIGHT|PORT_TEXT_BOTTOM);
```

If you want the text to be vertical in the middle center of the box use the following:

```c
LPort_SetTextAlignment(pPort,
    PORT_TEXT_MIDDLE|PORT_TEXT_CENTER|PORT_TEXT_VERTICAL);
```

`LPort_GetTextAlignment` and `LPort_SetTextAlignment` can be used together. For example, if you want to move the text to the bottom of the box without changing the horizontal placement or orientation you can use the following:

```c
int nAlignment = LPort_GetTextAlignment(pPort);  //remove horizontal placement leaving other attributes intact
nAlignment &= ~(PORT_TEXT_TOP|PORT_TEXT_CENTER|PORT_TEXT_BOTTOM);
//set horizontal placement to be the bottom of the box
nAlignment |= PORT_TEXT_BOTTOM;
LPort_SetTextAlignment(pPort, nAlignment);
```
Port Alignment Definitions

**horizontal placement**
- PORT_TEXT_LEFT
- PORT_TEXT_MIDDLE
- PORT_TEXT_RIGHT

**vertical placement**
- PORT_TEXT_TOP
- PORT_TEXT_CENTER
- PORT_TEXT_BOTTOM

**orientation**
- PORT_TEXT_HORIZONTAL
- PORT_TEXT_VERTICAL

Port text alignment is fully specified by an OR of one definition from each group.

Port text orientation is horizontal by default, therefore `PORT_TEXT_HORIZONTAL` is an optional definition.

By `PORT_TEXT_MIDDLE` we understand the horizontal middle and by `PORT_TEXT_CENTER` the vertical center, so that you can combine left with center but not left with middle or top with center.

For example:

- `(PORT_TEXT_LEFT|PORT_TEXT_TOP)` corresponds to horizontal text at the left top corner.
- `(PORT_TEXT_MIDDLE|PORT_TEXT_BOTTOM)` corresponds to horizontal text in the middle of the bottom.
- `(PORT_TEXT_LEFT|PORT_TEXT_CENTER|PORT_TEXT_VERTICAL)` corresponds to vertical text in the center at the left.
- `(PORT_TEXT_MIDDLE|PORT_TEXT_CENTER|PORT_TEXT_VERTICAL)` corresponds to vertical text centered in the middle of the port box.

**Version**

Available in L-Edit 8.3 and later versions.

**See Also**

“LPort_GetTextAlignment” (page 1468), “Port Alignment Definitions” on page 1471.
LPort_GetDefaultTextAlignment

    int LPort_GetDefaultTextAlignment( void );

Description

Gets saved text alignment that is used when new port is created.

Return Values

Returns an OR of port text alignment attributes if successful, -1 in case of failure. See “Port Alignment Definitions” on page 1471 for details.

Version

New with version 16.00 but also affects legacy ports.

See Also

“LPort_GetTextAlignment” (page 1468), “LPort_SetTextAlignment” (page 1470) “Port Alignment Definitions” on page 1471
LPort_SetDefaultTextAlignment

void LPort_SetDefaultTextAlignment( int nDefaultTextAlignment );

Description

Sets saved text alignment value that will be used when new port is created.

Version

New with version 16.00 but will also affect legacy ports.

See Also

“LPort_GetTextAlignment” (page 1468), “LPort_SetTextAlignment” (page 1470), “Port Alignment Definitions” on page 1471
**LLabel_New**

```c
LLabel LLabel_New( LCell cell, LLayer layer, const char* name, LCoord x0, LCoord y0 );
```

**Description**

Creates a new label in cell on layer with the specified text and anchor point coordinates \( x_0, y_0 \).

**Return Values**

Returns a pointer to the newly created label if successful; otherwise returns NULL.

**Parameters**

- `cell`  
  Cell that will contain the label.
- `layer`  
  Label layer.
- `text`  
  Label text string.
- `x0`  
  Lower left x-coordinate of label anchor.
- `y0`  
  Lower left y-coordinate of label anchor.

**Version Compatibility**

New with version 16.00.

**See Also**

“LLabel” (page 1754), “LTerminalType” (page 1799).
**LLabel_GetName**

```c
char* LLabel_GetName( LLabel label, char* name, const int maxlen );
```

**Description**

Gets the name of the specified label as a string (up to a maximum string length).

**Return Values**

Returns a pointer to the label name buffer if successful; otherwise returns NULL.

**Parameters**

- `instance`  
  Label whose name is to be retrieved.
- `name`  
  String (buffer) containing the name text.
- `maxlen`  
  Maximum length allowed for name.

**Version Compatibility**

New with version 16.00.

**See Also**

“LLabel” (page 1754), “LTerminalType” (page 1799).
**LLabel_SetName**

```c
LStatus LLabel_SetName(LCell cell, LLabel label, const char* name, LCoord textSize);
```

**Description**

Sets the name of the specified label in the specified cell.

**Return Values**

Returns LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `cell` Cell containing the label.
- `label` Label to be (re)named.
- `name` New name of label.
- `textSize` Label text size.

**Version Compatibility**

New with version 16.00.

**See Also**

“LLabel” (page 1754), “LTerminalType” (page 1799).
LPort_New_With_Object

LPort LPort_New_With_Object( LCell cell, const char* text, LObject object );

Description

Creates new port consisting of one given object.

Can be used as a first step (or the only step) to start creating non-standard ports (standard is one box and one label).

Return Values

Returns a pointer to the newly created port if successful; otherwise returns NULL.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>Cell that will contain the port.</td>
</tr>
<tr>
<td>text</td>
<td>Port text string.</td>
</tr>
<tr>
<td>object</td>
<td>Object to be linked with port. Can be any object except another port.</td>
</tr>
</tbody>
</table>

Version Compatibility

New with version 16.00.

See Also

“LLabel” (page 1754), “LTerminalType” (page 1799).
**LPort_GetObjectsList**

```c
LObject LPort_GetObjectsList( LPort port );
```

**Description**

Gets a pointer to the first object in a list of objects linked to the given port.

**Return Values**

Returns a pointer to the head of port's object list if successful; NULL otherwise.

**Parameters**

- **port**
  Specified port.

**Version Compatibility**

New with version 16.00.
LPort_GetObjectsNext

LObject LPort_GetObjectsNext( LPort port, LObject object );

Description

Gets a pointer to the object immediately following an object in the list of objects linked to the given port.

Return Values

Returns a pointer to the next object in the object list of a port if successful; otherwise returns NULL.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Specified port.</td>
</tr>
<tr>
<td>object</td>
<td>Specified object.</td>
</tr>
</tbody>
</table>

Version Compatibility

New with version 16.00.
LPort_AddObject

```c
LStatus LPort_AddObject( LCell cell, LPort port, LObject object );
```

Description

Adds given object to the port.

Return Values

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

Parameters

- `cell` Cell containing port and object.
- `port` Port to add object to.
- `object` Object to be added.

Version Compatibility

New with version 16.00.
LPort_RemoveObject

LStatus LPort_RemoveObject( LCell cell, LPort port, LObject object );

Description

Removes given object from the list of objects linked to the port.

Return Values

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

Parameters

- cell: Cell containing port and object.
- port: Port containing given object
- object: Object to be removed.

Version Compatibility

New with version 16.00.
**LObject_GetPort**

```
LStatus LPort_RemoveObject( LCell cell, LPort port,LObject object );
LPort LObject_GetPort( LCell cell, LObject object );
```

**Description**

Gets the port to which given object is linked.

**Return Values**

Pointer to the port contains the object or NULL.

**Parameters**

- `cell`  
  Cell that contains the object.
- `object`  
  Specified object.

**Version Compatibility**

New with version 16.00.
LPort_IsLegacy

LBoolean LPort_IsLegacy( LPort port );

Description

Determines if a port is a legacy port.

Return Values

Returns LTRUE if the port is a legacy port, LFALSE if not.

Parameters

port Specified port.

Version Compatibility

New with version 16.00.
LPort_DeletePortOnly

LStatus LPort_DeletePortOnly( LCell cell, LPort port );

Description

Deletes port leaving all linked objects intact.

Return Values

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

Parameters

<table>
<thead>
<tr>
<th>cell</th>
<th>Cell containing the port.</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Specified port.</td>
</tr>
</tbody>
</table>

Version Compatibility

New with version 16.00.
LPort_SetTerminalType

LStatus LPort_SetTerminalType( LCell cell, LPort port, LTerminalType nTerminalType );

Description

Sets the terminal type of a port.

Return Values

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

Parameters

- **cell**: Cell containing port and object.
- **port**: Port whose terminal type is being modified.
- **nTerminalType**: The value of new terminal type.

Version Compatibility

New with version 16.00.
LPort_GetTerminalType

LTerminalType LPort_GetTerminalType( LPort port );

Description

Gets the terminal type of a port.

Return Values

The terminal type of given port.

Parameters

port The specified port.

Version Compatibility

New with version 16.00.
LPort_SetAccessDirection

LStatus LPort_SetAccessDirection( LCell cell, LPort port, int nAccessDirection );

Description

Sets the access direction bits of a port.

Return Values

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.

Parameters

- **cell**: Cell containing the port.
- **port**: Port whose access direction bits being modified.
- **nAccessDirection**: The value of new access direction bits.

Version Compatibility

New with version 16.00.
LPort_GetAccessDirection

LStatus LPort_RemoveObject( LCell cell, LPort port, LObject object );
int LPort_GetAccessDirection( LPort port );

Description

Gets access direction bits of a port.

Return Values

Returns an OR of port access direction bits.

Parameters

port

The specified port.

Version Compatibility

New with version 16.00.
LPort_GetMustConnect

```c
char* LEDITAPI Upi_LPort_GetMustConnect(LPort pPort, char* szMustConnect,
const int cnMaxLen);
```

Description

Gets the MustConnect text of given port.

Return Values

Returns a pointer to the MustConnect text buffer if successful; otherwise returns NULL.

Parameters

- `pPort` The specified port.
- `szMustConnect` String (buffer) containing the MustConnect text of the port.
- `cnMaxLen` Maximum length allowed for szMustConnect.

Version Compatibility

New with version 16.00.
LPort_SetMustConnect

LStatus LEDITAPI Upi_LPort_SetMustConnect(LCell pCell, LPort pPort, const char* cszMustConnect) ;

Description

Sets the MustConnect text of given port.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

- **pCell**: Cell containing the port.
- **pPort**: The specified port.
- **cszMustConnect**: New value for the MustConnect text of the port.

Version Compatibility

New with version 16.00.
LPort_GetPinName

```c
char* Upi_LPort_GetPinName(LPort pPort, char* szPinName, const int cnMaxLen);
```

Description

Gets the Pin Name of given port.

Return Values

Returns a pointer to the Pin Name text buffer if successful; otherwise returns NULL.

Parameters

- **pPort**
  - The specified port.
- **szPinName**
  - String (buffer) containing the Pin Name of the port.
- **cnMaxLen**
  - Maximum length allowed for szPinName.

Version Compatibility

New with version 16.00.
**LLayer_FindLayerName**

```c
char* LLayer_FindLayerName( LFile pFile, int iLayerNumber, char* szName, const int iMaxLen );
```

**Description**

In a specified file searches the name of a layer with given layer number.

**Return Values**

Pointer to the layer name if found; NULL otherwise.

**Parameters**

- `pFile`  
  File whose layer list is to be searched.

- `iLayerNumber`  
  Layer number whose name is to be retrieved.

- `szName`  
  String (buffer) containing the name of the layer.

- `iMaxLen`  
  Maximum length allowed for szName.

**Version Compatibility**

New with version 16.00.
LLayer_FindPurposeName

```c
char* LLayer_FindPurposeName( LFile pFile, int iPurposeNumber, char* szPurposeName, const int iMaxLen );
```

**Description**

In a specified file searches for the name of a layer purpose with given purpose number.

**Return Values**

Pointer to the layer purpose name if found; NULL otherwise.

**Parameters**

- `pFile`: File whose layer list is to be searched.
- `iPurposeNumber`: Layer purpose number whose name is to be retrieved.
- `szPurposeName`: String (buffer) containing the name of the layer.
- `iMaxLen`: Maximum length allowed for `szPurposeName`.

**Version Compatibility**

New with version 16.00.
Selection Functions

Selected objects may be those selected with the mouse, those falling into a drawn box, all objects on a particular layer, or all objects of a particular cell. Once selected, they are entered into an internal selection list. Several functions may be applied to objects found in the selection list.

Selection functions allow the user to manipulate a selection in L-Edit.

- “LSelection_SelectAll” (page 1501)
- “LSelection_DeselectAll” (page 1502)
- “LSelection_AddObject” (page 1503)
- “LSelection_RemoveObject” (page 1504)
- “LSelection_Cut” (page 1495)
- “LSelection_Copy” (page 1498)
- “LSelection_SnapToMfgGrid” (page 1527)
- “LSelection_PasteToLayer” (page 1499)
- “LSelection_Clear” (page 1500)
- “LSelection_Duplicate” (page 1515)
- “LSelection_AddAllObjectsOnLayer” (page 1506)
- “LSelection_RemoveAllObjectsOnLayer” (page 1507)
- “LSelection_AddAllObjectsInRect” (page 1508)
- “LSelection_RemoveAllObjectsInRect” (page 1509)
- “LSelection_GetList” (page 1510)
- “LSelection_GetNext” (page 1511)
- “LSelection_GetLayer” (page 1512)
- “LSelection_ChangeLayer” (page 1513)
- “LSelection_Move” (page 1514)
- “LSelection_Flatten” (page 1518)
- “LSelection_Group” (page 1516)
- “LSelection_UnGroup” (page 1517)
- “LSelection_FlipHorizontal” (page 1520)
- “LSelection_FlipVertical” (page 1521)
- “LSelection_SliceHorizontal” (page 1522)
- “LSelection_SliceVertical” (page 1523)
- “LSelection_Rotate” (page 1524)
- “LSelection_RotateAroundPoint” (page 1525)
- “LSelection_Merge” (page 1519)
- “LSelection_GetObject” (page 1505)
LSelection_Cut

LStatus LSelection_Cut( void );

Description

Removes all objects in the selection and copies them into the paste buffer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

“LStatus” (page 1792), “Selection Functions” (page 1494)
LSelection_ConvertToCurvedPolygon

LStatus LSelection_ConvertToCurvedPolygon();

Description

If possible, converts selected object in active view to Polygon.

Return Values

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.
LSelection_ConnectedSegments

```
LStatus LSelection_ConnectSegments(LCoord nTolerance);
```

**Description**

If possible, connects segments for selected objects in active view.

**Return Values**

Returns L.StatusOK if successful. If an error occurs, L.Status contains the error value.
LSelection_Copy

LStatus LSelection_Copy(void);

Description

Copies all objects in the selection into the paste buffer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

“LStatus” (page 1792), “Selection Functions” (page 1494)
LSelection_PasteToLayer

LStatus LSelection_PasteToLayer( LLayer layer );

Description

Pastes the contents of the paste buffer to the given layer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

layer Layer on which paste buffer contents are to be pasted.

See Also

“LWireParam” (page 1808), “LStatus” (page 1792), “Selection Functions” (page 1494)
LSelection_Clear

LStatus LSelection_Clear( void );

Description

Removes all objects in the current selection.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

“LStatus” (page 1792), “Selection Functions” (page 1494)
LSelection_SelectAll

LSelection LSelection_SelectAll( void );

Description

Selects all objects in the current cell.

Return Values

Returns a pointer to the head of the selection list if successful; NULL otherwise.

See Also

“LSelection” (page 1788), “Selection Functions” (page 1494)
LSelection_DeselectAll

    void LSelection_DeselectAll( void );

Description

Deselects all objects in the current cell.

See Also

“Selection Functions” (page 1494)
LSelection_AddObject

\[
\text{LStatus LSelection_AddObject( LObject obj );}
\]

**Description**

Adds an object to the selection list.

**Return Values**

LStatusOK if successful. If an error occurs, \text{LStatus} contains the error value.

**Parameters**

\[
\text{obj}
\]

Object to be added to the selection list.

**See Also**

“LStatus” (page 1792), “LObject” (page 1769), “Selection Functions” (page 1494)
LSelection_RemoveObject

```c
LStatus LSelection_RemoveObject( LObject obj );
```

Description

Removes an object from the selection list.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

- `obj` Object to be removed from the selection list.

See Also

“LStatus” (page 1792), “LObject” (page 1769), “Selection Functions” (page 1494)
**LSelection_GetObject**

```c
LObject LSelection_GetObject( LSelection selection );
```

**Description**

Gets the object associated with a selection element.

**Return Values**

Pointer to the selection object if successful; NULL otherwise.

**Parameters**

`selection`  
Pointer to the selection element.

**See Also**

“LObject” (page 1769), “LSelection” (page 1788), “Selection Functions” (page 1494)
LSelection_AddAllObjectsOnLayer

LStatus LSelection_AddAllObjectsOnLayer( LLayer layer );

Description

Adds all objects on layer to the selection list.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

layer Layer whose objects are to be added to the selection.

See Also

“LStatus” (page 1792), “LWireParam” (page 1808), “Selection Functions” (page 1494)
**LSelection_RemoveAllObjectsOnLayer**

```c
LStatus LSelection_RemoveAllObjectsOnLayer( LLayer layer );
```

**Description**

Removes all objects on `layer` from the selection list.

**Return Values**

`LStatusOK` if successful. If an error occurs, `LStatus` contains the error value.

**Parameters**

`layer`
Layer whose objects are to be removed from the selection.

**See Also**

“`LStatus`” (page 1792), “`LWireParam`” (page 1808), “`Selection Functions`” (page 1494)
LSelection_AddAllObjectsInRect

LSelection_AddAllObjectsInRect( LRect *box );

Description

Adds all objects in rectangle box to the selection list.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

box Pointer to an LRect that specifies the coordinates of the box.

See Also

“LStatus” (page 1792), “LTransform” (page 1801), “Selection Functions” (page 1494)
**LSelection_RemoveAllObjectsInRect**

```c
LStatus LSelection_RemoveAllObjectsInRect( LRect *box );
```

**Description**

Removes all objects in rectangle `box` from the selection list.

**Return Values**

LStatusOK if successful. If an error occurs, `LStatus` contains the error value.

**Parameters**

`box`  
Pointer to an `LRect` that specifies the coordinates of the box.

**See Also**

“LStatus” (page 1792), “LTransform” (page 1801), “Selection Functions” (page 1494)
LSelection_GetList

LSelection  LSelection_GetList( void );

Description

Gets the pointer to the first element in the selection list.

Return Values

Pointer to the head of the selection list if successful; NULL otherwise.

See Also

“LSelection” (page 1788), “Selection Functions” (page 1494)
LSelection_GetNext

LSelection LSelection_GetNext( LSelection selection );

Description

Gets a pointer to the next element in the selection list.

Return Values

Pointer to the next element in the selection list if successful; NULL otherwise.

Parameters

selection Pointer to a selection element.

See Also

“LSelection” (page 1788), “Selection Functions” (page 1494)
LSelection_GetLayer

```c
LLabel LSelection_GetLayer( LSelection selection );
```

Description

Gets the layer of a given selection element.

Return Values

Pointer to the layer if successful; NULL otherwise.

Parameters

`selection`  
Pointer to the selection element.

See Also

“LWireParam” (page 1808), “LSelection” (page 1788), “Selection Functions” (page 1494)
LSelection_ChangeLayer

LStatus LSelection_ChangeLayer( LLayer srcLayer, LLayer dstLayer );

Description

If srcLayer is NULL, changes the layer of all objects in the selection to dstLayer. If srcLayer is not NULL, only selected objects that are originally on srcLayer will be changed to dstLayer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

srcLayer Source layer. NULL is a valid entry.
dstLayer Destination layer.

See Also

“LStatus” (page 1792), “LWireParam” (page 1808), “Selection Functions” (page 1494)
LSelection_Move

LStatus LSelection_Move( long dx, long dy );

Description

Moves the selection by displacements $dx$ (in the x-direction) and $dy$ (in the y-direction).

Return Values

LStatusOK if successful. If an error occurs, \texttt{LStatus} contains the error value.

Parameters

$dx$ Displacement value in x-direction.
$dy$ Displacement value in y-direction.

See Also

“LStatus” (page 1792), “Selection Functions” (page 1494)
LSelection_Duplicate

LStatus LSelection_Duplicate( void );

Description

Duplicates the contents of the current selection. The duplicate is placed next to the original.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

“LStatus” (page 1792), “Selection Functions” (page 1494)
**LSelection_Group**

```c
LStatus LSelection_Group( char *group_cell_name );
```

**Description**

Creates a new cell containing the currently selected objects and an instance of the new cell in the current cell.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `group_cell_name` Name of the group cell.

**See Also**

“LStatus” (page 1792), “Selection Functions” (page 1494)
**LSelection_UnGroup**

```c
LStatus LSelection_UnGroup( void );
```

**Description**

Ungroups (flattens one level of hierarchy of) the current selection.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**See Also**

“LStatus” (page 1792), “Selection Functions” (page 1494)
LSelection_Flatten

LStatus LSelection_Flatten( void );

Description

Flattens all levels of hierarchy (down to basic objects) in the current selection.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

“LStatus” (page 1792), “Selection Functions” (page 1494)
**LSelection_Merge**

```c
LStatus LSelection_Merge( void );
```

**Description**

Merges all objects in the current selection which share the same layer. If Quiet mode is ON, then the merge command will automatically combine all properties of the merged objects without notification.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**See Also**

“LStatus” (page 1792), “Selection Functions” (page 1494)
LSelection_FlipHorizontal

LStatus LSelection_FlipHorizontal( void );

Description

Flips all objects in current selection horizontally (left/right).

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

“LStatus” (page 1792), “Selection Functions” (page 1494)
LSelection_FlipVertical

LStatus LSelection_FlipVertical( void );

Description

Flips all objects in current selection vertically (up/down).

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

“LStatus” (page 1792), “Selection Functions” (page 1494)
LSelection_SliceHorizontal

LStatus LSelection_SliceHorizontal( LPoint *point );

Description

LSelection_SliceHorizontal slices horizontally all objects in current selection at the specified point.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

point Pointer to an LPoint structure that contains an (x,y) point on the horizontal slice line.

See Also

“LStatus” (page 1792), “LTransform” (page 1801), “Selection Functions” (page 1494)
LSelection_SliceVertical

LStatus LSelection_SliceVertical( LPoint *point );

Description

Slices vertically all objects in current selection at the specified point.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

point Pointer to an LPoint structure that contains an (x,y) point on the vertical slice line.

See Also

“LStatus” (page 1792), “LTransform” (page 1801), “Selection Functions” (page 1494)
LSelection_Rotate

LStatus LSelection_Rotate( void );

Description

Rotates all objects in current selection counterclockwise by 90 degrees.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

See Also

“LStatus” (page 1792), “Selection Functions” (page 1494)
LSelection_RotateAroundPoint

LStatus LSelection_RotateAroundPoint( double angle,
LCoord x, LCoord y, LBoolean bRelativeToCenter );

Description

Rotates the selected objects counterclockwise by the specified \textit{angle} (in degrees) with respect to the indicated reference point \((x, y)\). The reference point can be specified in absolute coordinates or as relative distances from the selection’s center point. If the value of \textit{angle} has more resolution that 6 decimal places, it will be rounded to 6 decimals places.

Return Values

Returns \texttt{LStatusOK} if successful. If an error occurs, \texttt{LStatus} contains the error type \texttt{LBadParameters}. An error will occur under any of the following conditions:

- \(|\textit{angle}| \geq 360\)
- \(|x| \geq \text{WORLD\_MAX}\)
- \(|y| \geq \text{WORLD\_MAX}\)
- \(b\text{RelativeToCenter}\) is invalid

Parameters

- \textit{angle} \quad The rotation angle in degrees. A positive angle indicates counterclockwise rotation. Rotated boxes become polygons when the specified angle is not orthogonal (a multiple of 90°).
- \(x, y\) \quad x- and y-coordinates of the specified center of rotation. May be given relative to the origin or to the selection’s center point.
- \(b\text{RelativeToCenter}\) \quad If \texttt{LTRUE}, indicates that the coordinates \((x, y)\) are given relative to the selection’s center point. If \texttt{LFALSE}, \((x, y)\) is interpreted relative to the cell’s origin (i.e., in absolute coordinates).

Example

/* Rotate 45° counter-clockwise around the selection center */
LSelection_RotateAroundPoint(45, 0, 0, LTRUE);

/* Rotate 22.5° clockwise around the point (10, 20) */
LSelection_RotateAroundPoint(-22.5, 10, 20, LFALSE);

Version

Available in L-Edit 8.4 and later versions.
See Also

**LSelection_SnapToMfgGrid**

```c
LStatus LSelection_SnapToMfgGrid ( void );
```

**Description**

Snaps all objects that are currently selected in the active layout view to the manufacturing grid. This is the same as the command *Draw > Convert > Snap to Manufacturing Grid*.

**Return Values**

- **LStatusOK** if successful. If an error occurs, `LStatus` contains the error value with possible values:
  - **LBadParameters**—The active view is not a layout view.
  - **LNoSelection**—No objects are selected.

**Example**

```c
LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    LSelection_SnapToMfgGrid();
}
```

**Version**

Available in L-Edit v13.1 and later versions.
Layer Functions

There are three categories of UPI layer functions.

“Design Layer Functions” (page 1529) allow the user to assign resistance or capacitance values or wire setup information to a layer. These functions also allow the user to make a layer hidden or visible in a display.

“Generated Layer Functions” (page 1558) allow the user to manipulate layers generated from other layers according to equations defined by the user.

“Layer Rendering Functions” (page 1575) are used to edit the information that defines how L-Edit displays a design layer.
Design Layer Functions

L-Edit supports an unlimited number of design layers. Layers may be assigned a capacitance value, a resistance value, and wire setup information. Layers may also be hidden or visible in the display.

Design layer functions allow the user to create and manipulate design layers in a file.

“LLayer_New” (page 1530)  “LLayer_Delete” (page 1531)
“LLayer_Find” (page 1532)  “LLayer_FindGDS” (page 1533)
“LLayer_GetList” (page 1534)  “LLayer_GetNext” (page 1535)
“LLayer_PrecedingLayer” (page 1536)  “LLayer_PrecedingLayerEx99” (page 1537)
“LLayer_GetName” (page 1538)  “LLayer_SetName” (page 1539)
“LLayer_GetParameters” (page 1540)  “LLayer_GetParametersEx830” (page 1541)
“LLayer_SetParameters” (page 1544)  “LLayer_SetParametersEx830” (page 1545)
“LLayer_GetCap” (page 1548)  “LLayer_SetCap” (page 1549)
“LLayer_GetRho” (page 1550)  “LLayer_SetRho” (page 1551)
“LLayer_GetParameters” (page 1540)  “LLayer_GetParametersEx830” (page 1541)
“LLayer_MoveLayer” (page 1556)  “LLayer_Copy” (page 1557)
**LLayer_New**

```c
LStatus LLayer_New( LFile file, LLayer layer, char *name );
```

**Description**

Creates a new layer in the specified file. All layers in a file are arranged in a list. The newly created layer is added to the layer list directly after the specified layer.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `file` File where new layer is to be added.
- `layer` Layer after which the new layer is to be added.
- `name` Name of the new layer.

**See Also**

**LLayer_Delete**

```c
LStatus LLayer_Delete( LFile file, LLayer layer );
```

**Description**

Deletes the specified layer from the specified file.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `file` : File containing the layer.
- `layer` : Layer to be deleted.

**See Also**

LLayer_Find

\[
\text{LLayer} \; \text{LLayer\_Find} \left( \text{LFile} \; \text{file}, \; \text{const} \; \text{char}* \; \text{name} \right);
\]

Description

Searches the layer list of the specified file for a layer with the given name.

Return Values

Pointer to the matching layer if successful; NULL otherwise.

Parameters

- \textit{file} : File whose layer list is to be searched.
- \textit{name} : Layer name to look for.

Example

The following example searches for the layer named Metal1 in the file layout.tdb:

```c
/*This example opens layout.tdb and checks to see if it contains a layer called Metal1*/

LFile file;
LLayer layer;

/*Open layout.tdb file*/
file = LFile\_Open("layout", TdbFile);

/*Search for layer Metal1 in this file*/
layer = LLayer\_Find(file, "Metal1");

if (layer == NULL){
    /*Layer not found*/
} else {
    /*layer found*/
}
```

The above example will return an opaque pointer layer to the layer Metal1. It thus saves the time required to write code for browsing through all the layers using LLayer\_GetList and LLayer\_GetNext.

See Also

“\textbf{LFile}” (page 1736), “\textbf{LWireParam}” (page 1808), “\textbf{Design Layer Functions}” (page 1529)
**LLayer_FindGDS**

```c
LLayer LLayer_FindGDS( LFile file, long gdsnum, long datatype);
```

**Description**

Searches the layer list of the specified file for a layer with the given GDSII number.

**Return Values**

Pointer to the matching layer if successful; NULL otherwise.

**Parameters**

- `file`: File whose layer list is to be searched.
- `gdsnum`: GDS layer number for which to search.
- `datatype`: Datatype for which to search.

**See Also**

LLayer_GetList

```
LLayer LLayer_GetList( LFile file );
```

**Description**

Gets a pointer to the first layer in the layer list of file.

**Return Values**

Pointer to the head of the layer list if successful; NULL otherwise.

**Parameters**

- `file` Specified file.

**See Also**

LLayer_GetNext

```c
LLayer LLayer_GetNext( LLayer layer );
```

**Description**

Gets a pointer to the layer immediately following a given layer in the layer list.

**Return Values**

Pointer to the next element in the layer list if successful; NULL otherwise.

**Parameters**

- `layer` Specified layer.

**See Also**

“LWireParam” (page 1808), “Design Layer Functions” (page 1529)
**LLayer_PrecedingLayer**

```c
LLayer LLayer_PrecedingLayer( LFile pFile, char* szName, LLayer pReserved);
```

**Description**

Finds the layer that precedes the specified layer’s name. Argument `pReserved` should be set to NULL when calling this function.

**Return Values**

Pointer to the preceding layer if successful; NULL otherwise.

**Parameters**

- `pFile` File whose layers are to be searched.
- `szName` Name of the layer whose preceding layer is required.
- `pReserved` Reserved variable. Set to NULL when calling this function.

**See Also**

**LLayer_PrecedingLayerEx99**

```c
LLayer LLayer_PrecedingLayerEx99( LFile pFile, LLayer pLayer );
```

**Description**

Finds the layer that precedes the specified layer.

**Return Values**

Pointer to the preceding layer if successful; NULL otherwise.

**Parameters**

- **pFile**
  - File whose layers are to be searched.
- **pLayer**
  - Specified layer.

**See Also**

“**LWireParam**” (page 1808), “**LFile**” (page 1736), “**Design Layer Functions**” (page 1529)
**LLayer GetName**

```c
char* LLayerGetName( LLayer layer, char* name, const int maxlen );
```

**Description**

Gets the name of a layer and fills the buffer `name` with the name of the layer. If the layer name is longer than `maxlen`, the extra characters are ignored.

**Return Values**

Pointer to the layer name buffer if successful; NULL otherwise.

**Parameters**

- `layer` : Layer whose name is to be retrieved.
- `name` : String (buffer) containing the name of the layer.
- `maxlen` : Maximum length allowed for `name`.

**Example**

```c
LFile pFile = LFile_GetVisible();
if ( pFile )
{
    LLayer pLayer = LLayer_GetList( pFile );
    char layernamemax[ MAX_LAYER_NAME ];
    if ( LLayerGetName( pLayer, layernamemax, sizeof( layernamemax ) ) )
        LDialog_MsgBox( layernamemax ); // print out first layer
}
```

**See Also**

“LWireParam” (page 1808), “Design Layer Functions” (page 1529)
LLayer_SetName

LStatus LLayer_SetName( LLayer layer, const char *name );

Description

Changes the name of a layer.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

layer
Layer whose name is to be changed.

name
String (buffer) that contains the new name of the layer.

See Also

LLayer_GetParameters

LStatus *LLayer_GetParameters( LLayer layer, LLayerParam *param );

Description

Gets the properties of layer.

Return Values

Pointer to the layer parameter structure if successful; NULL otherwise.

Parameters

layer
Specified layer.

param
Pointer to a layer parameter structure. This structure will be used for returning data.

See Also

LLayer_GetParametersEx830

```c
LStatus LLayer_GetParametersEx830( LLayer pLayer,
       LLayerParamEx830 *pLayerParam)
```

**Description**

Retrieves the parameters of a layer such as CIF name, GDSII layer number, GDSII layer data type, area and fringe capacitance, resistivity, default wire parameters, and lock and hidden states.

**Return Values**

*LStatusOK* if successful. If an error occurs, *LStatus* contains the error value with possible values:

- **LBadLayer**—*pLayer* is NULL
- **LBadParameters**—*pLayerParam* is NULL

**Parameters**

- **pLayer** Specified layer.
- **pLayerParameters** Pointer to an extended layer parameter structure. This structure will be used for returning data.

**Version**

This command is available in L-Edit V8.3 and later. It is also available as LLayer_GetParameters_Ex00().

**Example**

```c
double dAreaCap;
LFile pFile = LFile_GetVisible();
if(Assigned(pFile))
{
    LLayer pLayer = LLayer_Find(pFile, "Poly");
    if(Assigned(pLayer))
    {
        LLayerParam_Ex00 LayerParameters;
        if(LLayer_GetParameters_Ex00(pLayer, &LayerParameters) == LStatusOK)
        {
            dAreaCap = LayerParameters.AreaCapacitance;
            // More Processing
            // ...
        }
    }
}
```
Version

Available in L-Edit 8.2 and later versions.

See Also

**LLayer_GetParametersEx1512**

```c
LStatus LLayer_GetParametersEx830( LLayer pLayer, LLayerParamEx1512 *pLayerParam)
```

**Description**

Retrieves the parameters of a layer such as CIF name, GDSII layer number, GDSII layer data type, area and fringe capacitance, resistivity, default wire parameters, and lock and hidden states, and also the “protected” state.

**Return Values**

- **LStatusOK** if successful. If an error occurs, **LStatus** contains the error value with possible values:
  - **LBadLayer**—*pLayer* is NULL
  - **LBadParameters**—*pLayerParam* is NULL

**Parameters**

- **pLayer**
  - Specified layer.
- **pLayerParameters**
  - Pointer to an extended layer parameter structure. This structure will be used for returning data.

**Version**

Available in L-Edit version 15.12 and subsequent versions.

**See Also**

LLayer_SetParameters

```c
LStatus LLayer_SetParameters(LLayer layer, LLayerParam *param);
```

**Description**

Sets the parameters of `layer`.

**Return Values**

LStatusOK if successful. If an error occurs, `LStatus` contains the error value.

**Parameters**

- `layer` Specified layer.
- `param` Pointer to a layer parameter structure containing the new layer parameters.

**See Also**

“LStatus” (page 1792), “LWireParam” (page 1808), “LSpecialLayer” (page 1791),
“Design Layer Functions” (page 1529)
**LLayer_SetParametersEx830**

```c
LStatus LLayer_SetParametersEx830(LLayer pLayer, LLayerParamEx830 *pLayerParam);
```

**Description**

Sets the parameters of a layer such as CIF name, GDSII layer number, GDSII layer data type, area and fringe capacitance, resistivity, default wire parameters, and lock and hidden states.

**Return Values**

`LStatusOK` if successful. If an error occurs, `LStatus` contains the error value with possible values:

- **LBadLayer**—`pLayer` is NULL.
- **LBadParameters**—`LLayer_SetParametersEx830` returns `LBadParameters` when the `AreaCapacitance` or `FringeCapacitance` or `Resistivity` value in the structure specified by `pLayerParam` is invalid. One or more of the following error values is returned:
  - `pLayerParam` is NULL.
  - `AreaCapacitance` is invalid (this happens when the `pLayerParam` specified by `AreaCapacitance` < 0 and the `pLayerParam` specified by `AreaCapacitance` ≠ -1).
  - `FringeCapacitance` is invalid (this happens when the `pLayerParam` specified by `FringeCapacitance` < 0 and the `pLayerParam` specified by `FringeCapacitance` ≠ -1).
  - Resistivity is negative (`pLayerParam` specified by `Resistivity` < 0).
  - Invalid wire parameters are found.

**Parameters**

- `pLayer` Specified layer.
- `pLayerParam` Pointer to an extended layer parameter structure containing the new layer parameters.

**Example**

```c
LFile pFile = LFile_GetVisible();
if(Assigned(pFile)) {
    LLayer pLayer = LLayer_Find(pFile, "Poly");
    if(Assigned(pLayer)) {
        LLayerParamEx830 LayerParameters;
        if(LLayer_GetParametersEx830(pLayer, &LayerParameters) == LStatusOK) {
            LayerParameters.AreaCapacitance = 500.3;
            if(LLayer_SetParametersEx830(pLayer, &LayerParameters) == LStatusOK)
```
Version

Available in L-Edit 8.2 and later versions. It is also available as LLayer_SetParameters_Ex00().

See Also

LLayer_SetParametersEx1512

```c
LStatus LLayer_SetParametersEx1512(LLayer pLayer, LLayerParamEx1512 *pLayerParam);
```

**Description**

Sets the parameters of a layer, with the addition of the “protected” state.

**Parameters**

- `layer`: Specified layer.
- `param`: Pointer to a layer parameter structure. This structure will be used for returning data.

**Version**

Available in L-Edit 15.12 and subsequent versions.

**See Also**

“LLayerParamEx830” (page 1757), “LLayer_SetParametersEx830” (page 1545)
LLayer_GetCap

    double LLayer_GetCap( LLayer layer );

Description

    Gets the capacitance of layer.

Return Values

    The capacitance value of layer. It returns -1 on error.

Parameters

    layer                Specified layer.

See Also

    "LWireParam" (page 1808), "Design Layer Functions" (page 1529)
LLayer_SetCap

```c
LStatus LLayer_SetCap( LLayer layer, double cap );
```

Description

Changes the capacitance value of `layer`.

Return Values

LStatusOK if successful. If an error occurs, `LStatus` contains the error value.

Parameters

- `layer`: Specified layer.
- `cap`: Capacitance value.

See Also

LLayer_GetRho

```c
double LLayer_GetRho( LLayer layer );
```

**Description**

Gets the resistance value of `layer`.

**Return Values**

The resistance value of `layer`. It returns -1 on error.

**Parameters**

- `layer`: Specified layer.

**See Also**

“LWireParam” (page 1808), “Design Layer Functions” (page 1529)
**LLayer_SetRho**

```c
LStatus LLayer_SetRho( LLayer pLayer, double dRho );
```

**Description**

Changes the resistance of `pLayer`.

**Return Values**

LStatusOK if successful. If an error occurs, `LStatus` contains the error value.

**Parameters**

- `pLayer` Specified layer.
- `dRho` New resistance value.

**See Also**

LLayer_GetCurrent

LLayer LLayer_GetCurrent( LFile file );

Description

Gets a pointer to the current layer in the specified file.

Return Values

Pointer to the current layer if successful; NULL otherwise.

Parameters

file Specified file.

See Also

LLayer_SetCurrent

```c
LStatus LLayer_SetCurrent( LFile file, LLayer layer );
```

**Description**

Sets the current layer in the specified file.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `file` Specified file.
- `layer` Specified layer.

**See Also**

LLayer_GetSpecial

```c
LLayer LLayer_GetSpecial( LFile file, LSpecialLayer specialLayer );
```

Description

Gets a particular type of special layer of a file. Each file employs seven layers for specific purposes involved with graphic display. Each special layer is assigned a layer selected from the file’s layer list.

Return Values

Pointer to the special layer if successful; NULL otherwise.

Parameters

- `file` Specified file.
- `specialLayer` Type of special layer.

See Also

**LLayer_SetSpecial**

```c
LStatus LLayer_SetSpecial( LFile file, LSpecialLayer specialLayer, LLayer layer );
```

**Description**

Sets a special layer of a given file to a particular type. Each file employs seven layers for specific purposes involved with graphic display. Each special layer is assigned a layer selected from the file’s layer list. (For more information on special layers, see “Special Layers” on page 99.)

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `file` Specified file.
- `specialLayer` Type of special layer.
- `layer` New special layer.

**See Also**

**LLayer_MoveLayer**

```c
LStatus LLayer_MoveLayer( LFile pTDBFile, LLayer pLayerToMove, LLayer pLayerNewLocation );
```
LLayer_Copy

LLayer LLayer_Copy( LLayer pLayer );
Generated Layer Functions

Generated layers are generated from other layers according to an equation defined by the user. Generated layer definitions are assigned to each layer in the layer list and can be directly accessed and modified with the function calls below.

“LLayer_GetDerivedList” (page 1559)  
“LLayer_IsDerived” (page 1561)  
“LLayer_EnableAllDerived” (page 1562)  
“LLayer_GetDerivedParameters” (page 1564)  
“LLayer_GetDerivedParametersEx830” (page 1565)  
“LLayer_DestroyDerivedParameter” (page 1568)  
“LCell_GenerateLayers” (page 1571)  
“LLayer_GetDerivedNext” (page 1560)  
“LLayer_DisableAllDerived” (page 1563)  
“LLayer_SetDerivedParameters” (page 1566)  
“LLayer_SetDerivedParametersEx830” (page 1567)  
“LLayer_DestroyDerivedParameterEx840” (page 1570)  
“LCell_ClearGenerateLayers” (page 1574)
LLayer_GetDerivedList

LLayer LLayer_GetDerivedList( LFile file );

Description

Gets the list of generated layers in a file.

Return Values

Pointer to the head of the generated layer list if successful; NULL otherwise.

Parameters

file

Specified file.

See Also

“LWireParam” (page 1808), “LFile” (page 1736), “Generated Layer Functions” (page 1558)


**LLayer_GetDerivedNext**

```c
LLayer LLayer_GetDerivedNext( LLayer layer );
```

**Description**

Gets the generated layer following a given generated layer.

**Return Values**

Pointer to the next element in the generated layer list if successful; NULL otherwise.

**Parameters**

- `layer` Specified layer.

**See Also**

“LWireParam” (page 1808), “Generated Layer Functions” (page 1558)
LLayer_IsDerived

```c
int LLayer_IsDerived( LLayer layer );
```

**Description**

Checks whether a layer is a generated layer or not.

**Return Values**

A nonzero value if the layer is a generated layer, or zero if the layer is not a generated layer.

**Parameters**

- `layer` Specified layer.

**See Also**

“LWireParam” (page 1808), “Generated Layer Functions” (page 1558)
LLayer_EnableAllDerived

\[ \text{LStatus} \ \text{LLayer_EnableAllDerived}( \text{LFile} \ \text{file}) ; \]

Description

Enables the generated layer definition for all layers in a file.

Return Values

\text{LStatusOK} if successful. If an error occurs, \text{LStatus} contains the error value.

Parameters

\text{file} \quad \text{Specified file.}

See Also

“\text{LStatus}” (page 1792), “\text{LFile}” (page 1736), “\text{Generated Layer Functions}” (page 1558)
**LLayer_DisableAllDerived**

```c
LStatus LLayer_DisableAllDerived( LFile file );
```

**Description**

Disables the generated layer definition for all layers in the specified file.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

`file` Specified file.

**See Also**

“LStatus” (page 1792), “LFile” (page 1736), “Generated Layer Functions” (page 1558)
LLayer_GetDerivedParameters

```c
LDerivedLayerParam* *LLayer_GetDerivedParameters( LLabel layer,
                                                LDerivedLayerParam *param );
```

**Description**

Gets the parameters of a generated layer.

**Return Values**

Pointer to the generated layer parameter structure if successful; NULL otherwise.

**Note:**

Note that this function is superseded by “LLayer_GetDerivedParametersEx830” (page 1565).

**Parameters**

- `layer` Specified layer.
- `param` Pointer to a generated layer parameter structure.

**See Also**

“LSpecialLayer” (page 1791), “LWireParam” (page 1808), “Generated Layer Functions” (page 1558),
“LLayer_GetDerivedParametersEx830” (page 1565).
LLayer_GetDerivedParametersEx830

```c
LLayer_GetDerivedParametersEx830* LLayer_GetDerivedParametersEx830(LLabel layer,
                             LDerivedLayerParamEx830 *param);
```

**Note:**
LLayer_GetDerivedParametersEx830 can be used interchangeably with LLayer_GetDerivedParametersEx00.

**Description**

Gets derivation parameters of the specified layer into the LDerivedLayerParamEx00 structure pointed to by the specified parameter value.

**Note:**
Note that this function is superseded in L-Edit V10 and later.

**See Also**

“LStipple” (page 1793), “LDerivedLayerAreaOperation” (page 1711),
LLayer_SetDerivedParameters

```c
LStatus LLayer_SetDerivedParameters(LFile file, LLabel layer, LDerivedLayerParam *param);
```

**Description**

Sets the generated layer parameters of a layer in a given file.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Note:**

Note that this function is superseded by “LLayer_GetParametersEx830” (page 1541).

**Parameters**

- `file` File containing the specified layer.
- `layer` Specified layer.
- `param` Pointer to a generated layer parameters structure that contains the new parameter values.

**See Also**

“Generated Layer Functions” (page 1558).
**LLayer_SetDerivedParametersEx830**

```c
LStatus LLayer_SetDerivedParametersEx830(LFile file, LLabel layer,
LDerivedLayerParamEx830 *param)
```

**Note:**
LLayer_SetDerivedParametersEx830 can be used interchangeably with LLayer_SetDerivedParametersEx00.

**Description**

Sets derivation parameters of the specified layer to the values specified in the `LDerivedLayerParam` structure pointed to by the specified parameter value.

**Note:**
Note that this function supersedes LLayer_SetDerivedParameters as of v8.30.

**See Also**

“LStipple” (page 1793), “LDerivedLayerAreaOperation” (page 1711),
**LLayer_DestroyDerivedParameter**

```c
LStatus LLayer_DestroyDerivedParameter(LLayerDerivedParam* pDerivedLayerParam)
```

**Description**

Frees the memory associated with the derived layer parameter structure that was allocated by L-Edit during an `LLayer_GetDerivedParameters` call.

**Note:** Do not call `Layer_DestroyDerivedParameter` if `LDrcRule_GetParameter` has not been previously called with `pDesignRuleParam`.

**Return Values**

- `LStatusOK` if successful. If an error occurs, `LStatus` contains the error value with possible values:
  - `LBadParameters` — `pDerivedLayerParam` is NULL

**Example**

```c
LFile pTDBFile = LFile_GetVisible();
if(Assigned(pTDBFile))
{
   LLayer pLayer = LLayer_Find(pTDBFile, "PolyCnt_And_NotPoly");
   if(Assigned(pLayer))
   {
      LDerivedLayerParam pDerivedLayerParam;
      if(Assigned(LLayer_GetDerivedParameters(pLayer, &pDerivedLayerParam)))
      {
         long lGrow = pDerivedLayerParam.layer1_grow_amount;
         // More Processing
         // ...
         LLayer_DestroyDerivedParameter(&pDerivedLayerParam);
      }
   } // endif(Assigned(pLayer))
} // endif(Assigned(pTDBFile))
```

**Version**

Available in L-Edit 8.2 and later versions.

**Note:** Note that this function is superseded in L-Edit v10 and later by `LLayer_DestroyDerivedParameterEx840`. 
See Also

“LStatus” (page 1792), “LDerivedLayerParam” (page 1716), “Generated Layer Functions” (page 1558)
LLayer_DestroyDerivedParameterEx840

LStatus LLayer_DestroyDerivedParameterEx840(
    LDerivedLayerParamEx830 *pDerivedLayerParam );
**LCell_GenerateLayers**

```c
LStatus LCell_GenerateLayers( LCell cell, int bin_size );
```

**Description**

Generates layers in the specified cell.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- **cell**
  Specified cell.
- **bin_size**
  Bin size.

**Note:**

Note that this function is obsolete in L-Edit V10 and later.

**Version**

**Note:**

As of L-Edit version 10.00 this function is superseded by LCell_GenerateLayers_v11_10 and LCell_GenerateLayers_v10_00.

**See Also**

**LCell_GenerateLayers_v11_10**

```c
LStatus LCell_GenerateLayers_v11_10( LCell pcell, const char* szCommandFile,
const char** pszArrayOfLayerNames, unsigned int nNumberOfLayers, LBoolean
bMergeObjectsAfterGeneration);
```

**Description**

Generates layers in the specified cell.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

**See Also**

“LCell_GenerateLayers_v10_00” (page 1573), “Generated Layer Functions” (page 1558).
**LCell_GenerateLayers_v10_00**

```c
LStatus LCell_GenerateLayers_v10_00( LCell pCell, LLayer* ArrayOfLayers,
    unsigned int nNumberOfLayers, LBoolean bClearAllGeneratedLayers, LBoolean
    bMergeObjectsAfterGeneration );
```

**Description**

Generates layers in the specified cell.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

**See Also**

“LCell_GenerateLayers_v11_10” (page 1572), “Generated Layer Functions” (page 1558)
**LCell_ClearGenerateLayers**

```c
LStatus LCell_ClearGenerateLayers( LCell cell );
```

**Description**

Clears all generated layers from a cell.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `cell` Specified cell.

**See Also**

Layer Rendering Functions

The rendering pass list contains the layer rendering information that L-Edit uses to display a layer. Information found in this list include stipple pattern, color, pass type and write mode (set or clear). (For additional information, see “Rendering Layer Parameters” on page 93.)

L-Edit UPI recognizes two fixed values related to rendering. You can use these constants to construct loops that step through all possible outline styles or rendering attributes:

- **NumberOfOutlineStyles**
  - Number of outline styles defined.

- **NumberOfRenderingAttributes**
  - Number of rendering attributes per layer.

- "**LLayer_GetRenderingAttribute**“ (page 1581)
- "**LLayer_SetRenderingAttribute**“ (page 1582)
- "**LLayer_GetRenderingObjectName**“ (page 1583)
- "**LFile_GetColorPalette**“ (page 1584)
- "**LFile_GetColorPaletteNumColors**“ (page 1585)
- "**LFile_GetColorPaletteSortBy**“ (page 1586)
- "**LFile_SetColorPalette**“ (page 1587)
- "**LFile_SetColorPaletteNumColors**“ (page 1588)
- "**LFile_SetColorPaletteSortBy**“ (page 1589)

**Obsolete:**

- "**LPass_New**“ (page 1576)
- "**LPass_GetList**“ (page 1577)
- "**LPass_GetNext**“ (page 1578)
- "**LPass_GetParameters**“ (page 1579)
- "**LPass_SetParameters**“ (page 1580)
LPass_New

LPass LPass_New( LPass precedingPass, LPass pass );

Description

Adds a new pass after the preceding pass.

Return Values

Pointer to the newly added pass if successful; NULL otherwise.

Parameters

precedingPass  
Preceding pass. The new pass will be added after precedingPass.

pass  
Pass to be deleted.

Note:  
Note that this function is superseded in L-Edit V10 and later.

See Also

“LPass” (page 1775), “Layer Rendering Functions” (page 1575)
**LPass_GetList**

```c
LPass LPass_GetList( LLabel layer, LPassType passType );
```

**Description**

Gets a list of particular pass types associated with a layer.

**Return Values**

Pointer to the head of the pass list if successful; NULL otherwise.

**Parameters**

- `layer` Specified layer.
- `passType` Type of pass.

**Note:** Note that this function is superseded in L-Edit V10 and later.

**See Also**

LPass_GetNext

LPass LPass_GetNext( LPass pass );

Description

Gets the next pass in the pass list.

Return Values

Pointer to the next element in the pass list if successful; NULL otherwise.

Parameters

pass Specified pass.

Note: Note that this function is superseded in L-Edit V10 and later.

See Also

“LPass” (page 1775), “Layer Rendering Functions” (page 1575)
LPass_GetParameters

LPParam *LPass_GetParameters( LPass pass, LPParam *param );

Description

Gets the parameters of a pass.

Return Values

Pointer to the pass parameters structure if successful; NULL otherwise.

Parameters

- *pass: Specified pass.
- *param: Pointer to a pass structure. This buffer will be filled with the results.

*Note:* Note that this function is superseded in L-Edit V10 and later.

See Also

**LPass_SetParameters**

```c
LStatus LPass_SetParameters( LPass pass, LPassParam *param );
```

**Description**

Sets the parameters of a pass.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `pass` Specified pass.
- `param` Pointer to a pass structure. This buffer contains the new parameter values.

**Note:**

Note that this function is superseded in L-Edit V10 and later.

**See Also**

LLayer_GetRenderingAttribute

```c
LStatus LLayer_GetRenderingAttribute( LLabel layer,
          LRenderingAttributeIndex index, LRenderingAttribute *pRA );
```

Description

This function returns a rendering attribute.

Return Values

LStatusOK if successful or LBadParameters if not.

Parameters

- `layer` The layer
- `index` The number of the rendering attribute to get.
- `pRA` A pointer to `LRenderingAttribute` structure.

Example

```c
unsigned int get_port_text_pass(LLayer layer)
{
    LRenderingAttribute ra;
    LLayer_GetRenderingAttribute( layer, raiPortText, &ra );
    return ra.mPass;
}
```

See Also

- “LStipple” (page 1793), “LRenderingAttribute” (page 1784),
- “LLayer_GetRenderingObjectName” (page 1583), “LLayer_SetRenderingAttribute” (page 1582)
LLayer_SetRenderingAttribute

```c
LStatus LLayer_SetRenderingAttribute( LLabel layer,
  LRenderingAttributeIndex index, LRenderingAttribute *pRA );
```

Description

This function sets a rendering attribute.

Return Values

LStatusOK if successful or LBadParameters if not.

Parameters

- **layer** — The layer
- **index** — The number of the rendering attribute to set.
- **pRA** — A pointer to LRenderingAttribute structure.

Example

```c
void make_outline_thin(LLayer layer)
{
  unsigned int n;
  LRenderingAttribute ra;

  for(n=raiFirstRenderingAttribute; n<=raiLastRenderingAttribute; n++)
  {
    LLayer_GetRenderingAttribute(layer, n; &ra);
    ra.mOutlineThicknessUnits = utPixels;
    ra.mOutlineThickness = 1;
    LLayer_SetRenderingAttribute(layer, n; &ra);
  }
}
```

See Also

“LStipple” (page 1793), “LRenderingAttribute” (page 1784),
“LLayer_GetRenderingObjectName” (page 1583), “LLayer_GetSpecial” (page 1554),
“LLayer_SetSpecial” (page 1555)
LLayer_GetRenderingObjectName

LStatus LLayer_GetRenderingObjectName( LLabel layer, LRenderingAttributeIndex index, char *nameBuf, int nameBufSize );

Description

This function is mainly for debugging purposes. It returns the name of a rendering attribute.

Return Values

LStatusOK if successful or LBadParameters if not. The possible values of nameBuf after a successful call are: “Object,” “PortBox,” “PortText,” “WireCenterline,” “SelectedObject,” “SelectedPortBox,” “SelectedPortText,” and “SelectedWireCenterline.”

Parameters

layer The layer
index The number of the rendering attribute to get.
nameBuf The buffer that will contain the rendering attribute name
nameBufSize Maximum number of characters to put into nameBuf

Example

void message_outline_thickness(LLayer layer)
{
    unsigned int n;
    LRenderingAttribute ra;
    char nameBuf[64];
    char msgBuf[NumberOfRenderingAttributes][128];

    for(n=raiFirstRenderingAttribute; n<=raiLastRenderingAttribute; n++)
    {
        LLayer_GetRenderingObjectName(layer, n, nameBuf, sizeof(nameBuf));
        LLayer_GetRenderingAttribute(layer, n; &ra);

        sprintf(msgbuf[n], "Outline thickness for %s is %u %s", 
                nameBuf, 
                ra.mOutlineThickness, 
                (ra.mOutlineThicknessUnits==utPixels)? "Pixels" : "LU");
    }

    LDialog_MultiLineMsgBox(msgBuf, NumberOfRenderingAttributes);
}

See Also

“LRenderingAttribute” (page 1784), “LLayer_GetRenderingObjectName” (page 1583), “LLayer_SetRenderingAttribute” (page 1582)
LFile_GetColorPalette

LStatus LEDITAPI LFile_GetColorPalette( LFile file, LColor *pColor, int index );
LFile_GetColorPaletteNumColors

int LEDITAPI LFile_GetColorPaletteNumColors( LFile file );
LFile_GetColorPaletteSortBy

const char* LEDITAPI LFile_GetColorPaletteSortBy( LFile file );
LFile_SetColorPalette

LStatus LEDITAPI LFile_SetColorPalette(LFile file, const LColor *pColor, int index);
LFile_SetColorPaletteNumColors

LStatus LEDITAPI LFile_SetColorPaletteNumColors( LFile file, int numcolors );
LFile_SetColorPaletteSortBy

LStatus  LEDITAPI LFile_SetColorPaletteSortBy( LFile file, const char *sortby );
Technology Setup Functions

Technology functions allow the user to manipulate the technology of a design file. Specifically, these functions allow the user to get, set, or change the technology setup or individual technology parameters.

“LFile_GetTechnology” (page 1591)
“LFile_SetTechnology” (page 1592)
“LFile_SetTechnologyName” (page 1593)
“LFile_SetTechnologyUnitNum” (page 1594)
“LFile_SetTechnologyUnitDenom” (page 1595)
“LFile_SetTechnologyUnitName” (page 1596)
“LFile_SetTechnologyLambdaNum” (page 1597)
“LFile_SetTechnologyLambdaDenom” (page 1598)
“LFile_GetTechnologyEx840” (page 1599)
“LFile_SetTechnologyEx840” (page 1600)
LFile_GetTechnology

LTechnology LFile_GetTechnology( LFile file );

Description

Gets the technology setup of a file.

Return Values

The LTechnology structure filled with the values of the current technology setup.

Parameters

file Specified file.

Note: Note that this function is superseded in L-Edit V10 and later.

See Also

**LFile_SetTechnology**

```c
LStatus LFile_SetTechnology( LFile file, LTechnology *technology );
```

**Description**

Sets the technology setup of a file.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `file` Specified file.
- `technology` Pointer to an LTechnology structure that contains the new technology setup.

**Note:**

Note that this function is superseded in L-Edit V10 and later.

**See Also**

LFile_SetTechnologyName

    char* LFile_SetTechnologyName( LFile file, char* name );

Description

Sets the technology name of file.

Return Values

Pointer to the technology name buffer if successful; NULL otherwise.

Parameters

    file          Specified file.
    name          New technology name.

See Also

“LFile” (page 1736), “Technology Setup Functions” (page 1590)
**LFile_SetTechnologyUnitNum**

```c
LStatus LFile_SetTechnologyUnitNum( LFile file, long num );
```

**Description**

Sets the numerator of the technology unit mapping fraction in file to `num`.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `file` Specified file.
- `num` Numerator value.

**See Also**

**LFile_SetTechnologyUnitDenom**

```c
LStatus LFile_SetTechnologyUnitDenom( LFile file, long denom );
```

**Description**

Sets the denominator of the technology unit mapping fraction in file to `denom`.

**Return Values**

- LStatusOK if successful. If an error occurs, `LStatus` contains the error value.

**Parameters**

- `file`: Specified file.
- `denom`: Denominator value.

**See Also**

**LFile_SetTechnologyUnitName**

\[
\text{LStatus LFile_SetTechnologyUnitName( LFile file, const char* name );}
\]
LFile_SetTechnologyLambdaNum

\begin{verbatim}
LStatus LFile_SetTechnologyLambdaNum(LFile file, long num);
\end{verbatim}

**Description**

Sets the numerator of the technology lambda mapping fraction in file to \textit{num}.

**Return Values**

LStatusOK if successful. If an error occurs, \textit{LStatus} contains the error value.

**Parameters**

- \textit{file}  
  Specified file.
- \textit{num}  
  Numerator value.

**See Also**

LFile_SetTechnologyLambdaDenom

LStatus LFile_SetTechnologyLambdaDenom( LFile file, long denom );

Description

Sets the denominator of the technology lambda mapping fraction in file to denom.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Specified file.</td>
</tr>
<tr>
<td>denom</td>
<td>Denominator value.</td>
</tr>
</tbody>
</table>

See Also

LFile_GetTechnologyEx840

```c
LStatus LFile_GetTechnologyEx840( LFile pTDBFile, LTechnologyEx840 *pTechnology );
```

**Description**

Gets the technology setup of a file.

**Return Values**

The `LTechnologyEx840` structure filled with the values of the current technology setup.

**Parameters**

- `pTDBFile` Specified file.
- `pTechnology` pointer to a `LTechnologyEx840` structure

**See Also**

LFile_SetTechnologyEx840

LStatus LFile_SetTechnologyEx840( LFile pTDBFile, LTTechnologyEx840 *pTechnology );

Description

Sets the technology setup of a file.

Return Values

LStatus OK if successful. If an error occurs, LStatus contains the error value.

Parameters

  pTDBFile Specified file.
  pTechnology Pointer to an LTTechnologyEx840 structure that contains the new technology setup.

See Also

“LStatus” (page 1792), “LFile” (page 1736), “LTTechnologyEx840” (page 1798),
“Technology Setup Functions” (page 1590)
Color Palette Functions

L-Edit color palette can contain 16, 32, 64, 128, or 256 colors. (For further information, see “Setting the Number of Available Colors” on page 60.)

These functions allow the user to manipulate the color palette of a layout.

“LFile_GetColorPalette” (page 1602)  “LFile_SetColorPalette” (page 1605)
“LFile_GetColorPaletteNumColors” (page 1603)  “LFile_SetColorPaletteNumColors” (page 1606)
“LFile_GetColorPaletteSortBy” (page 1604)  “LFile_SetColorPaletteSortBy” (page 1607)
**LFile_GetColorPalette**

```c
LStatus LFile_GetColorPalette( LFile file, LColor *pColor, int index );
```

**Description**

Gets a color from the palette.

**Return Values**

Returns LStatusOK if successful or LBadParameter if an error occurred.

**Parameters**

- `file` Current file.
- `pColor` Pointer to `LColor`.
- `index` Index number of the color to get. Must be non-negative and less than the number of colors in the palette.

**Structure**

```c
typedef structure {
    short LRed
    short LBlue
    short LGreen
} LColor
```

**See Also**

“LFile_SetColorPalette” (page 1605), “LFile_GetColorPaletteNumColors” (page 1603)
**LFile_GetColorPaletteNumColors**

```c
int LFile_GetColorPaletteNumColors( LFile file );
```

**Description**

Gets the number of colors in the palette.

**Return Values**

Number of colors in the palette. Possible values are:

- 16
- 32
- 64
- 128
- 256

Returns null if there is an error.

**Parameters**

- `file` Current file.

**See Also**

“LFile_SetColorPalette” (page 1605), “LFile_SetColorPaletteNumColors” (page 1606)
LFile_GetColorPaletteSortBy

const char *LFile_GetColorPaletteSortBy( LFile file );

Description

Sets the name of the palette sort option.

Return Values

The name of the palette sort option. Possible values are:

- "SortByIndex"
- "SortByNumBits"
- "SortByHue"
- "SortByBrightness"

Returns null if an error occurred.

Parameters

file Current file.

See Also

“LFile_SetColorPaletteSortBy” (page 1607)
LFile_SetColorPalette

```
LStatus LFile_SetColorPalette( LFile file, const LColor *pcolor, int index );
```

**Description**

Sets a color specified by the index in the palette.

**Return Values**

Returns LStatusOK if successful or LBadParameter if an error occurred.

**Parameters**

- **file**
  - Current file.
- **pcolor**
  - Pointer to LColor.
- **index**
  - Index of the color to set. Must be non-negative and less than the number of colors in the palette.

**Structure**

```
typedef structure {
    short LRed
    short LBlue
    short LGreen
} LColor
```

**See Also**

**LFile_SetColorPaletteNumColors**

```c
LStatus LFile_SetColorPaletteNumColors( LFile file, int numcolors );
```

**Description**

Sets the number of colors in the palette. This number must be one of the following values:

- 16
- 32
- 64
- 128
- 256

**Return Values**

Returns LStatusOK if successful or LBadParameter if an error occurred.

**Parameters**

- `file` Current file.

**See Also**

“LFile_GetColorPaletteSortBy” (page 1604)
LFile_SetColorPaletteSortBy

LStatus LFile_SetColorPaletteSortBy( LFile file, const char *sortby );

Description

Sets a name of the palette sort option. Possible values are:

- "SortByIndex"
- "SortByNumBits"
- "SortByHue"
- "SortByBrightness"

Return Values

Returns LStatusOK if successful or LBadParameter if an error occurred.

Parameters

file Current file.

See Also

“LFile_GetColorPaletteSortBy” (page 1604).
Import/Export Functions

L-Edit can import a layout from GDS II and CIF files or export a layout to GDS II or CIF files.

“CIF Setup Functions” (page 1609) allow the user to set CIF import/export parameters.

“GDSII Setup Functions” (page 1613) allow the user to set GDS II import/export parameters.

“DRC Functions” (page 1622) allow the user to manipulate the design rules of a layout file and run a design rule check.

“Extract Functions” (page 1643) are used for netlist extraction.

“Core Functions” (page 1656) allow the user to manipulate the core of a layout file.
CIF Setup Functions

“LFile_GetCIFParameters” (page 1610)  “LFile_SetCIFParameters” (page 1611)
**LFile_GetCIFParameters**

```c
LCIFParam *LFile_GetCIFParameters( LFile file, LCIFParam *cifparam);
```

**Description**

Gets the CIF parameters of a file.

**Return Values**

Pointer to the CIF parameters structure if successful; NULL otherwise.

**Parameters**

- `file` Specified file.
- `cifparam` Pointer to a structure that will contain CIF parameters.

**See Also**

“LGDSParam” (page 1738), “LFile” (page 1736), “CIF Setup Functions” (page 1609)
LFile_SetCIFParameters

LStatus LFile_SetCIFParameters( LFile file, LCIFParam *cifparam );

Description

Sets the CIF parameters of a file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

file Specified file.

cifparam Pointer to a structure that contains the new CIF parameter values.

See Also

**LFile_ImportCIF**

```c
LStatus LFile_ImportCIF ( LFile pFile, const char* pcszSrcFile, LBoolean bPolygonAsRect, LOverwriteCellsScopeOnImport OverwriteCells, const char* pcszLogFileName );
```

**Description**

Imports and merges the given CIF file with the given design. Logging is optional and will occur only if a log file name is provided. The log file will not open in the user interface.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- **pFile**
  - Design file to import to.

- **pcszSrcFile**
  - Source CIF file to import from.

- **bPolygonAsRect**
  - Read rectangular polygons as boxes.

- **OverwriteCells**
  - Overwrite option. Overwrite all cells, Top Design Cells only, or Don't overwrite any existing cell.

- **pcszLogFileName**
  - Log will be written into this file if given. NULL - if don't want logging.
GDSII Setup Functions

- “LFile_GetGDSParameters” (page 1614)*
- “LFile_SetGDSParameters” (page 1616)*
- “LFile_ExportGDSII” (page 1618)
- “LGDSParam” (page 1738)

**Note:**

*These functions are superseded as of L-Edit v10 by “LGDSParam” (page 1738).
LFile_GetGDSParameters

LGDSParam *LFile_GetGDSParameters( LFile file, LGDSParam *gdsparam );

Version

Note: Note that this function was available in L-Edit versions 8.2 through 10. It is superseded in L-Edit v10 and later by “LGDSParam” (page 1738).

Description

Gets GDSII parameters of a file.

Return Values

Pointer to the GDSII parameters structure if successful; LBadParameters if pGDSParam or pfile is null.

Parameters

file Specified file.

* gdsparam Pointer to a structure that will contain GDS II parameters.

Example 1

LGDSParam Params;
LFile_GetGDSParameters(File, &Params);
Params.use_default_units=true;
LFile_SetGDSParameters(File, &Params);

Example 2

/****************************************************************************
 ****
 * Macro Name: Macro1
 * Creator :
 *
 * Revision History:
 * 6 May 2011Generated by L-Edit
 */
#include <cstdlib>
#include <cstdarg>
#include <cstdio>
#include <cstring>
#include <cctype>
#include <cmath>
#include <string>
// #define EXCLUDE.LEDIT_LEGACY_UPI // <-- This should be commented out in v15.12

#include <ldata.h>
/* Begin -- Uncomment this block if you are using L-Comp. */
// #include <lcomp.h>
/* End */

extern "C"
{
  void Macro1(void);
  int UPI_Entry_Point(void);
}

void Macro1(void)
{
  LCell pCell = LCell_GetVisible();
  LFile pFile = LCell_GetFile(pCell);

  // TODO: Insert your code here
  LGDSParam Params;
  LFile_GetGDSParameters(pFile, &Params);
  Params.use_default_units=false;
  LFile_SetGDSParameters(pFile, &Params);
}

int UPI_Entry_Point(void)
{
  LMacro_BindToMenuAndHotKey_v9_30("Tools", NULL /*hotkey*/,
                                    "Macro1", "Macro1", NULL /*hotkey category*/);
  return 1;
}

See Also

LFile_SetGDSParameters

LStatus LFile_SetGDSParameters( LFile file, LGDSParam *gdsparam );

Description

Sets the current GDSII parameters of a file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value LBadParameters if gdsparam or file is null.

Parameters

**file**
Specified file.

**gdsparam**
Pointer to a structure that contains GDSII parameters.

See Also

LFile ImportGDSII

**LStatus LFile ImportGDSII**( LFile pFile, const char* pcszSrcFile, LBoolean bUseGDSDataType, LOverwriteCells ScopeOnImport OverwriteCells, LBoolean bGdsFileResolution, double dCustomResolution, const char* pcszLogFileName);

**Description**

Imports and merges the specified GDSII file with the active design.

**Return Values**

Returns LStatusOK if successful, otherwise returns LStatus with an error code.

**Parameters**

- **pFile**
  Design file to import to.

- **pcszSrcFile**
  Source GDSII file to import from.

- **bUseGDSDataType**
  Treat different GDSII data types on a layer as different layers.

- **OverwriteCells**
  Overwrite option. Overwrite All Cells, Top Design Cells only, or Don't overwrite any existing cell.

- **bGdsFileResolution**
  Resolution option. Use the resolution from source GDSII file or use custom resolution provided in dCustomResolution.

- **dCustomResolution**
  Custom resolution value.

- **pcszLogFileName**
  Log will be written into this file if given. Leave blank if you don't want logging.

**See Also**

Chapter 34: UPI Functions Reference

LFile_ExportGDSII
LStatus LFile_ExportGDSII( LFile file, LGDSParamEx *gdsparam,
LGDSExportLogParams* gdsExportLogParam );

Description
Exports GDSII to a given destination file.

Return Values
Returns LStatusOK if successful, otherwise returns an error code.

Parameters
file

Specified file.

gdsparam

A pointer to LGDSParamEx structure with specified export
parameters.

gdsExportLogParam

A pointer to LGDSExportLogParams structure with specified logging
parameters. Can be NULL if no logging is required.

Example
char szCellName[MAX_CELL_NAME];
LCell_GetName(pCell, szCellName, MAX_CELL_NAME);
char szLibName[MAX_CELL_NAME];
LFile_GetTopLevelLibraryName(pFile, szLibName, MAX_CELL_NAME);
char szLogFileName[512];
char szGDSFileName[512];
char szPath[512];
if (LStatusOK == LFile_GetLibraryPath(pFile, szLibName, szPath, 512))
{
char *pch = strrchr(szPath, '\\');
if(pch)
{
int index = pch - szPath + 1;
strncpy(szLogFileName, szPath, index);
szLogFileName[index] = '\0';
}
strcat(szLogFileName, szCellName);
strcat(szGDSFileName, szLogFileName);
strcat(szLogFileName, ".log");
strcat(szGDSFileName, ".gds");
}
LGDSParamEx GdsParam;
GdsParam.cszDestFileName = szGDSFileName;
GdsParam.bZipOutputFile = LFALSE;
// Export scope value enumaration. See below for values.
GdsParam.ExportScope = gdsExportActiveCell;

L-Edit 16 User Guide

Section 3

1618


/Contains Cell Name if ExportScope is gdsExportSpecifiedCell
GdsParam.cszSpecifiedCell = NULL;
// Libraries to include ExportScope is gdsExportCellsFromLibraries
GdsParam.cpszIncludeLibraries = NULL;
// LTRUE if hierarchy of specified cells need to be included in the export
// or LFALSE.
GdsParam.bIncludeHierarchy = LTRUE;
// List of libraries to exclude (empty or NULL if none)
GdsParam.cpszExcludeLibraries = NULL;
// Use default units
GdsParam.bUseDefaultUnits = LTRUE;
// Following two values specify custom Units if bUserDefaultUnits is LFALSE
GdsParam.dMicrons = 0;
GdsParam.dUserUnits = 0;
// 0 for PFile::GDS_CasePreserve, 1 for PFile::GDS_CaseUpper,
// 2 for PFile::GDS_CaseLower
GdsParam.nUpcaseCellName = 0;
// Restrict cell name character count
GdsParam.nCellNameLength = 32; // 32 - Standard
// Map file name
GdsParam.cszMapFileName = NULL;
GdsParam.bDoNotExportHiddenObjects = LTRUE; //Do not export hidden objects
// and layers
GdsParam.bOverwriteGDSIIDataType = LFALSE; //Overwrite object data type with
// layer data type
GdsParam.bCalcChecksum = LTRUE; //Calculate MOSIS checksum
GdsParam.bCheckSelfIntersections = LTRUE; //Check self-intersecting polygons
// and wires
GdsParam.bFracture = LFALSE; //Fracture polygons with more than
// nFractureLimit vertices
GdsParam.nFractureLimit = 199;

LGDSExportLogParams GdsExportLogParam;
GdsExportLogParam.szLogFileName = szLogFileName;
GdsExportLogParam.bOpenLogInWindow = LFALSE;

if (LStatusOK == LFile_ExportGDSII(pFile, &GdsParam, &GdsExportLogParam))
{
    LUpi_LogMessage(LFormat("Successfully exported cell %s to: ",
        szCellName));
    LUpi_LogMessage(LFormat("%s\n", GdsParam.cszDestFileName));
}

See Also

LFile_ImportTechnology_GDSLayerMap

```c
LStatus LFile_ImportTechnology_GDSLayerMap( LFile file, const char* pcszSrcFile );
```

**Description**

Imports a GDS layermap file into the specified database. Assigns GDS layer names and datatypes to layer-purpose pairs based on layermap file.

**Return Values**

Returns LStatusOK if successful, otherwise returns an error code.

**Parameters**

- `file`  
  Pointer to file into which GDS layer map is imported
- `pcszSrcFile`  
  GDS layer map file
LFile_ImportTechnology_LEF

LStatus LFile_ImportTechnology_LEF( LFile file, const char* pcszSrcFile, const char* pcszTargetLibrary, LBoolean bImportLayerResistancesAndCapacitances );

Description

Imports a GDS layermap file into the specified database. Assigns GDS layer names and datatypes to layer-purpose pairs based on layermap file.

Return Values

Returns LStatusOK if successful, otherwise returns an error code.

Parameters

- **file**: Pointer to file into which LEF file is imported
- **pcszSrcFile**: LEF file
- **pcszTargetLibrary**: Target Library name
- **bImportLayerResistancesAndCapacitances**: LTRUE to import layer resistances and capacitances.
DRC Functions

DRC functions allow the user to manipulate the design rules of a layout file and run a design rule check.

“LDrcRule_Add” (page 1623)  “LDrcRule_Delete” (page 1624)
“LDrcRule_GetList” (page 1626)  “LDrcRule_GetNext” (page 1627)
“LDrcRule_Find” (page 1625)  “LDrcRule_SetRuleSet” (page 1628)
“LDrcRule_GetParameters” (page 1630)  “LDrcRule_SetParameters” (page 1631)
“LDRCRule_DestroyParameter” (page 1632)  “LDrcRule_SetTolerance” (page 1629)
“LFile_GetDrcFlags” (page 1637)  “LFile_SetDrcFlags” (page 1638)
“LCell_OpenDRCSummary” (page 1639)  “LCell_OpenDRCStatistics” (page 1640)
“LCell_GetDRCNumErrors” (page 1641)  “LCell_GetDRCStatus” (page 1642)
“LCell_RunDRCEx01” (page 1305)

Obsolete:
“LDRC_Run” (page 1634)
“LFile_GetBinSize” (page 1635)  “LFile_SetBinSize” (page 1636)
LDrcRule_Add

```c
LDrcRule LDrcRule_Add( LFile file, LDrcRule preceding_rule, LDesignRuleParam *param );
```

Description

Adds a new design rule to the file. The newly added design rule will be added after the specified `preceding_rule` and will have the specified parameters.

Return Values

Pointer to the newly added DRC rule if successful; NULL otherwise.

Parameters

- **file**
  - Specified file.
- **preceding_rule**
  - New rule will be added after this rule.
- **param**
  - Pointer to a design rule parameter structure that specifies the details of the new rule.

See Also

“LGDSParam” (page 1738), “LFile” (page 1736), “DRC Functions” (page 1622)
LDrcRule_Delete

```c
LStatus LDrcRule_Delete( LFile file, LDrcRule rule );
```

**Description**

Deletes a design rule from a file.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `file` Specified file.
- `rule` Rule to be deleted.

**See Also**

LDrcRule_Find

LDrcRule LDrcRule_Find( LFile file, LDrcRuleType rule_type, char *layer1, char *layer2 );

Description

Searches for a specific design rule involving two given layers. If layer2 is the blank in the rule (as for metal1 to metal1 spacing), it should be set to NULL. For example, LDrcRule(File, LSPACING, layer, NULL).

Return Values

Pointer to the DRC rule if successful; NULL otherwise.

Parameters

file Specified file.
rule_type Type of DRC rule.
layer1 Source layer 1.
layer2 Source layer 2.

See Also

“LGDSParm” (page 1738), “LFile” (page 1736), “LDrcRuleType” (page 1727), “DRC Functions” (page 1622)
LDrcRule_GetList

LDrcRule LDrcRule_GetList( LFile file );

Description

Gets a list of DRC rules in a file.

Return Values

Pointer to the head of the DRC rule list if successful; NULL otherwise.

Parameters

file

Specified file.

See Also

“LGDSParam” (page 1738), “LFile” (page 1736), “DRC Functions” (page 1622)
LDrcRule_GetNext

LDrcRule LDrcRule_GetNext( LDrcRule rule );

Description

Gets the design rule that follows a given design rule.

Return Values

Pointer to the next element in the DRC rule list if successful; NULL otherwise.

Parameters

rule Specified design rule.

See Also

“LGDSParm” (page 1738), “DRC Functions” (page 1622)
LDrcRule_SetRuleSet

```c
LStatus LDrcRule_SetRuleSet( LFile file, char *rule_set );
```

Description

Sets the name of the design rule set in a file.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

- **file**
  Specified file.

- **rule_set**
  Name of the rule set.

See Also

LDrcRule_SetTolerance

```c
LStatus LDrcRule_SetTolerance( LFile file, long tolerance );
```

**Description**

Sets the tolerance of the design rule set in a file.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- `file` Specified file.
- `tolerance` Tolerance of the design rule set.

**See Also**

“LStatus” (page 1792), “LFile” (page 1736), “DRC Functions” (page 1622)
LDrcRule_GetParameters

```c
LDesignRuleParam *LDrcRule_GetParameters( LDrcRule rule,
LDesignRuleParam *param );
```

Description

Gets the parameters of a DRC rule.

Return Values

Pointer to the DRC rule parameters structure if successful; NULL otherwise.

Parameters

- `rule` Specified design rule.
- `param` Pointer to a structure that will contain the parameters.

See Also

“LGDSParm” (page 1738), “DRC Functions” (page 1622)
LDrcRule_SetParameters

```c
LStatus LDrcRule_SetParameters( LFile file, LDrcRule rule,
    LDesignRuleParam *param );
```

**Description**

Sets the parameters of a DRC rule.

**Return Values**

LStatusOK if successful. If an error occurs, LStatus contains the error value.

**Parameters**

- **file**
  Specified file.
- **rule**
  Specified design rule.
- **param**
  Pointer to a structure that contains the design rule parameters.

**See Also**

LDRCRule_DestroyParameter

LStatus LDrcRule_DestroyParameter( LDesignRuleParam *pDesignRuleParam );

Description

Frees the memory associated with the design rule parameter structure that was allocated by L-Edit during a LDrcRule_GetParameter call.

Note: Do not call LDrcRule_DestroyParameter if LDrcRule_GetParameter has not been previously called with pDesignRuleParam.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value with possible values:

LBadParameters — pDesignRuleParam is NULL

Parameters

pDesignRuleParam Pointer to a design rule parameter structure.

Example

LFile pTDBFile = LFile_GetVisible();
if(Assigned(pTDBFile))
{
    LDrcRule pDRCRule = LDrcRule_Find(pTDBFile, LSPACING, "Poly", NULL);
    if(Assigned(pDRCRule))
    {
        LDesignRuleParam pDesignRuleParameter;
        if(Assigned(LDrcRule_GetParameters(pDRCRule, &pDesignRuleParameter)))
        {
            long lDist = pDesignRuleParameter.distance;
            // More Processing
            // ...
            LDrcRule_DestroyParameter(&pDesignRuleParameter);
        }
    } // endif(Assigned(pDRCRule))
} // endif(Assigned(pTDBFile))

Version

Available in L-Edit 8.2 and later versions.
See Also

“DRC Functions” (page 1622), “LDesignRuleParam” (page 1722), “LStatus” (page 1792)
LDRC_Run

```c
void LDRC_Run( LCell inCell, LRect* onArea, char* errfile,
               int writeErrorPorts, int writeErrorObjects );
```

Description

Runs DRC on the specified area of a cell.

Parameters

- **cell**
  Cell on which DRC is to be run.

- **onArea**
  Pointer to a `LRect` structure that specifies a rectangle where DRC will be run.

- **errfile**
  Name of the error file.

- **writeErrorPorts**
  If 1, error ports will be drawn.

- **writeErrorObjects**
  If 1, error objects will be written to the output file.

**Note:** This function is superseded by “LCell_RunDRCEx01” (page 1305).

See Also

- “LObject” (page 1769), “LTransform” (page 1801), “DRC Functions” (page 1622)
LFile_GetBinSize

LCoord LFile_GetBinSize( LFile pFile );

Description

Gets the DRC bin size.

Return Values

The DRC bin size.

Parameters

pFile
A TDB file.

Note:
Note that this function is superseded in L-Edit V10 and later.

See Also

“LCoord” (page 1707), “LDRC_Run” (page 1634)
LFile_SetBinSize

```c
void LFile_SetBinSize( LFile pFile LCoord lBinSize);
```

**Description**

Sets the DRC bin size.

**Parameters**

- `pFile` A TDB file.
- `lBinSize` The bin size.

**Note:** Note that this function is superseded in L-Edit V10 and later.

**See Also**

“LCoord” (page 1707), “LDRC_Run” (page 1634)
LFile_GetDrcFlags

```c
LStatus LFile_GetDrcFlags(LFile file, LDrcFlags* pDrcFlags);
```
LFile_SetDrcFlags

LStatus LFile_SetDrcFlags( LFile file, const LDrcRule *pDrcFlags );
**LCell_OpenDRCSummary**

LWindow LCell_OpenDRCSummary( LCell pCell );

**Description**

LCell_OpenDRCSummary opens the DRC Summary Report in a new text window in L-Edit for the specified cell.

**Return Values**

A pointer to the text window with the DRC Summary Report.

**Parameters**

- **pCell** Pointer to the cell.

**Example**

```plaintext
LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    LCell_OpenDRCSummary(pCell)
}
```

**Version**

Available in L-Edit v11.0 and later versions.
**LCell_OpenDRCStatistics**

```c
LWindow LCell_OpenDRCStatistics( LCell pCell );
```

**Description**

LCell_OpenDRCStatistics opens the DRC Statistics Report in a new text window in L-Edit for the specified cell.

**Return Values**

A pointer to the text window with the DRC Statistics Report.

**Parameters**

- **pCell**
  
  Pointer to the cell.

**Example**

```c
LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    LCell_OpenDRCStatistics(pCell)
}
```

**Version**

Available in L-Edit v11.0 and later versions.
LCell_GetDRCNumErrors

unsigned int LCell_GetDRCNumErrors( LCell pCell );
LCell_GetDRCStatus

LDrcStatus LCell_GetDRCStatus( LCell pCell );
Extract Functions

These functions are used for netlist extraction.

“LExtract_Run” (page 1644)  “LCell_OpenExtractSummary” (page 1647)
“LExtract_Run_Dialog” (page 1645)

Obsolete:
“LExtract_RunEx840” (page 1649)
“LExtract_GetOptions_Ex98” (page 1651)  “LExtract_SetOptionsEx840” (page 1654)
“LExtract_GetOptionsEx840” (page 1652)  “LExtract_GetOptionsEx840” (page 1652)
LExtract_Run

LStatus LExtract_Run(LCell cell, char *extDefFile, char *spiceOutFile, int writeNodeName, int writeNodeCapacitance);

Description

Runs L-Edit/Extract on a given cell.

See Also

“LStatus” (page 1792), “LObject” (page 1769), “Extract Functions” (page 1643)
LExtract_Run_Dialog

LStatus LExtract_Run_Dialog(LCell topCell);

Description

Runs L-Edit/Extract on a given cell. Invokes the Extract dialog to perform the extract operation. Note that a layout view should be active as topCell is going to be displayed in it.

LExtract_Run_Dialog differs from LExtract_RunEx840 in that with LExtract_RunEx840 the extract options are passed as parameters but with LExtract_Run_Dialog a dialog is invoked for user input of the extract options.

Return Values

LStatusOK if successful, LBadParameters if there is no current layout view, LBadCell if topCell is not assigned.

Parameters

topcell

The cell on which to perform the extract operation.

Example

LCell Cell = LCell_GetVisible();
LExtract_Run_Dialog(Cell);

Version

Available in L-Edit 8.4 and later versions.

See Also

“LExtract_RunEx840” (page 1649), “LExtract_GetOptionsEx840” (page 1652), “LExtract_SetOptionsEx840” (page 1654)
**LExtract_RunCommandFile**

LStatus LExtract_RunCommandFile(LCell pTopCell, const char* szCommandFile, const char* szSpiceOutFile);

**Description**

LExtract_RunCommandFile runs HiPer using a Calibre, Assura or Dracula LVS command file or a standard Extract EXT file. It extracts the cell as a hierarchical netlist.

**Return Values**

- **LStatusOK**— if successful. If an error occurs, LStatus contains the error value with possible values:
  - **LBadCell**— pCell is NULL
  - **LOpenError**— szCommandFile is null.
  - **LCreateError**— szSpiceOutFile is null.
  - **LSystemError**— Extract didn’t run due to some problem (parsing error, crash, etc).
  - **LUserAbort**— The user canceled extract while it was running.
  - **LCopyProtViolation**— HiPer could not get a license.

**Parameters**

- **pCell**
  
  Pointer to the cell.

- **CommandFile**
  
  String with the path and filename of the HiPer command file. If this is an empty string "", then it will run the standard extract rules.

- **SpiceOutFile**
  
  String with the path and filename of the output SPICE output file.

**Example**

LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    if(LCell_RunDRCCommandFile(pCell, "C:\TSMC\TSMC_HiPer.cal", "Cell.sp") == LStatusOK)
    {
        // More Processing
        // ...
    }
}

**Example**

Available in L-Edit 13.0 and later versions.
**LCell_OpenExtractSummary**

```c
LWindow LCell_OpenExtractSummary(LCell pCell);
```

**Description**

LCell_OpenExtractSummary opens the Extract Summary Report in a new text window in L-Edit for the specified cell.

**Return Values**

A pointer to the text window with the Extract Summary Report.

**Parameters**

- `pCell` Pointer to the cell.

**Example**

```c
LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    LCell_OpenExtractSummary(pCell)
}
```

**Version**

Available in L-Edit v13.0 and later versions.
**LCell_OpenExtractStatistics**

```c
LWindow LCell_OpenExtractStatistics(LCell pCell);
```

**Description**

LCell_OpenExtractStatistics opens the Extract Statistics Report in a new text window in L-Edit for the specified cell.

**Return Values**

A pointer to the text window with the Extract Statistics Report.

**Parameters**

- `pCell` Pointer to the cell.

**Example**

```c
LCell pCell = LCell_GetVisible();
if(Assigned(pCell))
{
    LCell_OpenExtractStatistics(pCell)
}
```

**Version**

Available in L-Edit v13.0 and later versions.
LExtract_RunEx840

LStatus LExtract_RunEx840( LCell topCell, LExtractOptionsEx840 *ExtOptions );

Description

Runs Extract on the cell topCell, using the options specified in ExtOptions.

Return Values

Returns LStatusOK if successful. If an error occurs, LStatus contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBadCell</td>
<td>TopCell is NULL.</td>
</tr>
<tr>
<td>LBadParameters</td>
<td>Indicates one or more of the following errors:</td>
</tr>
<tr>
<td></td>
<td>▪ There is no current view.</td>
</tr>
<tr>
<td></td>
<td>▪ There is no read access to ExtOptions.szExtDefnFile.</td>
</tr>
<tr>
<td></td>
<td>▪ There is no write access to ExtOptions.szExtDefnFile.</td>
</tr>
<tr>
<td></td>
<td>▪ ExtOptions.dExtractBinSize ≥ 0.</td>
</tr>
<tr>
<td></td>
<td>▪ ExtOptions.ParasiticCutoff &lt; 0.</td>
</tr>
<tr>
<td></td>
<td>▪ Any of the specified layers in ExtOptions is not a valid layer.</td>
</tr>
</tbody>
</table>

Parameters

*topCell* The top cell on which to run Extract.

*ExtOptions* Pointer to the structure containing Extract options. If this pointer is NULL, then Extract is run using the most recently set option values.

Examples

/* If the Extract options have previously been set, you can simply call
LExtract_RunEx840: */
LExtract_RunEx840(MyCell, NULL);

/* Otherwise, allocate a structure for the Extract options*/
LExtractOptionsEx840 ExtOptions;
ExtOptions.nMaxIncludeStmtLen = 4096;
ExtOptions.pszExtIncludeStmt = new char[ExtOptions.nMaxIncludeStmtLen];

/* Get the Extract Options for MyCell */
LExtract_GetOptionsEx840(MyCell, &ExtOptions);

/* Change the output file name */
strcpy(ExtOptions.szExtOutFile, "c:\ExtractOutput.out");

/* Run Extract */
LExtract_RunEx840(MyCell, &ExtOptions);
delete [] ExtOptions.pszExtIncludeStmt;
ExtOptions.nMaxIncludeStmtLen = 0;

Version

Available in L-Edit 8.4 and later versions.

See Also

“LCell” (page 1700), “LExtractOptionsEx840” (page 1733), “LExtract_GetOptionsEx840” (page 1652),
“LExtract_SetOptionsEx840” (page 1654)
**LExtract_GetOptions_Ex98**

```
LStatus LExtract_GetOptions_Ex98( LCell oCell, LExtractOptions *ExtOptions );
```

**Description**

Retrieves the L-Edit/Extract options for the given cell (topcell). The resulting extract options are stored in ExtOptions.

**Note:**

To properly retrieve the `.include` statement, the data member `szExtIncludeStmt` of the structure `ExtOptions` must be dynamically allocated to a size big enough to hold the expected `.include` statement. The data member `lMaxIncludeStmtLen` must be set to the size of the allocated string `szExtIncludeStmt`. Failure to do so could result in a general protection fault.

**Return Values**

LStatusOK if successful. If an error occurs, `LStatus` contains the error value.

**Note:**

Note that this function is superseded in L-Edit V10 and later.

**See Also**

“LStatus” (page 1792), “LObject” (page 1769), “LTechnology” (page 1797)
**LExtract_GetOptionsEx840**

```c
LStatus LExtract_GetOptions_Ex840( LCell cell, LExtractOptionsEx840 *ExtOptions );
```

**Description**

Retrieves the L-Edit/Extract options for the given cell (`cell`). The resulting Extract options are stored in `ExtOptions`.

**Note:** In order to properly retrieve the `.include` statement, the data member `szExtIncludeStmt` of the structure `ExtOptions` must be dynamically allocated to a sufficient size. The data member `nMaxIncludeStmtLen` must be set to the size of the allocated string `szExtIncludeStmt`. Failure to do so could result in a general protection fault. See “LExtractOptionsEx840” (page 1733) for more information about this data structure.

**Return Values**

Returns **LStatusOK** if successful. If an error occurs, **LStatus** contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LBadCell</strong></td>
<td><code>cell</code> is NULL.</td>
</tr>
<tr>
<td><strong>LBadParameters</strong></td>
<td><code>ExtOptions</code> is NULL.</td>
</tr>
</tbody>
</table>

**Parameters**

- **cell**: The cell from which to read L-Edit/Extract options.
- **ExtOptions**: Destination for Extract options to be written.

**Examples**

```c
/* Allocate a structure for the Extract options */
LExtractOptionEx840 ExtOptions;
ExtOptions.nMaxIncludeStmtLen = 4096;
ExtOptions.pszExtIncludeStmt = new char[ExtOptions.nMaxIncludeStmtLen];

/* Write Extract options to the specified destination */
LExtract_GetOptionsEx840(MyCell, &ExtOptions);

/* Do some processing */
...

/* After processing, release the memory */
delete [] ExtOptions.pszExtIncludeStmt;
ExtOptions.nMaxIncludeStmtLen = 0;
```
Version

Available in L-Edit 8.4 and later versions.

See Also

**LExtract_SetOptionsEx840**

```c
LStatus LExtract_SetOptionsEx840( LCell cell, LExtractOptionsEx840 *ExtOptions );
```

**Description**

Sets Extract options in the specified `cell` using the values stored in `ExtOptions`.

**Return Values**

Returns `LStatusOK` if successful. If an error occurs, `LStatus` contains the error type with possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LBadCell</code></td>
<td><code>cell</code> is NULL.</td>
</tr>
<tr>
<td><code>LBadParameters</code></td>
<td><code>ExtOptions</code> is NULL.</td>
</tr>
</tbody>
</table>

**Parameters**

- `cell` The cell in which to set L-Edit/Extract options.
- `ExtOptions` Pointer to Extract options.

**Examples**

```c
/* Allocate a structure for the Extract options */
LExtractOptionsEx840 ExtOptions;
ExtOptions.nMaxIncludeStmtLen = 4096;
ExtOptions.pszExtIncludeStmt = new char[ExtOptions.nMaxIncludeStmtLen];

/* Get Extract Options from MyCell*/
LExtract_GetOptionsEx840(MyCell, &ExtOptions);

/* Change the output file name */
strcpy(ExtOptions.szExtOutFile, "c:\\ExtractOutput.out");

/* Set Extract options */
LExtract_SetOptionsEx840(MyCell, &ExtOptions);

/* Release memory */
delete [] ExtOptions.pszExtIncludeStmt;
ExtOptions.nMaxIncludeStmtLen = 0;
```

**Version**

Available in L-Edit 8.4 and later versions.
See Also

Core Functions

The core is the "heart" of the design, where the functional logic is contained. It may be one large block containing all of the logic for the design, or it may be composed of several smaller blocks, which typically each have different functions within the design.

Core functions allow the user to manipulate the core of a layout file. The first function provide a way to check if a core exists. The other functions allow the user to get or set the layer-to-layer capacitance for a design’s horizontal or vertical routing layer.

“LCore_GetCore” (page 1657)
“LCore_GetLLHCap” (page 1658)  “LCore_SetLLHCap” (page 1659)
“LCore_GetLLVCap” (page 1660)  “LCore_SetLLVCap” (page 1661)
ЛCore_GetCore

LCore LCore_GetCore( LFile file );

Description

Gets the core of the specified file.

Return Values

Pointer to the core if successful; NULL otherwise.

Parameters

file

Specified file.

See Also

“LTеchnology” (page 1797), “LFile” (page 1736), “Core Functions” (page 1656)
LCore_GetLLHCap

    double LCore_GetLLHCap( LCore core );

Description

    Gets the layer-to-layer capacitance for the horizontal routing layer of a core.

Return Values

    The capacitance value (in aF/sq. micron), or -1 on error.

Parameters

    core                Specified core.

See Also

    “LTechnology” (page 1797), “Core Functions” (page 1656)
**LCore_SetLLHCap**

```c
LStatus LCore_SetLLHCap( LCore core, double LLHCap );
```

**Description**

Sets the layer-to-layer capacitance for the horizontal routing layer of a core.

**Return Values**

LStatusOK if successful. If an error occurs, **LStatus** contains the error value.

**Parameters**

- `core` Specified core.
- `LLHCap` New capacitance value (in aF/sq. micron).

**See Also**

- “**LStatus**” (page 1792), “**LTechnology**” (page 1797), “**Core Functions**” (page 1656)
LCore_GetLLVCap

```c
double LCore_GetLLVCap( LCore core );
```

Description

Gets the layer-to-layer capacitance for the vertical routing layer of a core.

Return Values

The capacitance value (in aF/sq. micron), or -1 on error.

Parameters

- `core` Specified core.

See Also

“LTTechnology” (page 1797), “Core Functions” (page 1656)
LCore_SetLLVCap

LStatus LCore_SetLLVCap( LCore core, double LLVCap );

Description

Sets the layer-to-layer capacitance for the vertical routing layer of a core.

Return Values

LStatusOK if successful. If an error occurs, LStatus contains the error value.

Parameters

- **core**
  - Specified core.

- **LLVCap**
  - New capacitance value (in aF/sq. micron).

See Also

“LStatus” (page 1792), “LTechnology” (page 1797), “Core Functions” (page 1656)
Utility Functions

The categories of utility functions are as follows:

“Point Functions” (page 1663) allow the user to create or transform a point.

“Rectangle Functions” (page 1669) allow the user to create or transform a rectangle.

“Transformation Functions” (page 1673) allow the user to adjust the translation, orientation, or manipulation of an object.

“Cross Section Functions” (page 1683) allow the user to invoke and use the cross section dialog.

“T-Cell Functions” (page 1685) allow the user to set T-Cell parameters.
Point Functions

“LPoint_Set” (page 1664)
“LPoint_Add” (page 1665)          “LPoint_Subtract” (page 1666)
“LPoint_Transform_Ex99” (page 1668)

Obsolete:
“LPoint_Transform” (page 1667)
LPoint_Set

LPoint LPoint_Set( LCoord x, LCoord y );

Description

Creates an LPoint type from two LCoord types with the values x and y.

Return Values

Returns the newly created LPoint.

Parameters

x x-coordinate.

y y-coordinate.

See Also

“LTransform” (page 1801)
**LPoint_Add**

```c
LPoint LPoint_Add( LPoint ptA, LPoint ptB );
```

**Description**

Adds two points

**Return Values**

The resultant point.

**Parameters**

- `ptA` : Point 1.
- `ptB` : Point 2.

**See Also**

“**LPoint**” (page 1780), “**Point Functions**” (page 1663)
LPoint_Subtract

\[
\text{LPoint} \quad \text{LPoint\_Subtract} ( \text{LPoint} \ pta, \text{LPoint} \ ptb );
\]

Description

Subtracts two points

Return Values

The resultant point.

Parameters

\[
\begin{align*}
\text{pta} & \quad \text{Point 1.} \\
\text{ptb} & \quad \text{Point 2.}
\end{align*}
\]

See Also

“Point Functions” (page 1663)
LPoint_Transform

LPoint LPoint_Transform( LPoint point, LTransform transform );

Description

Applies transform to a point.

Return Values

Values of a new point. The original point is not modified.

Parameters

point Specified point.
transform Specified transformation.

Note: Note that this function is superseded in L-Edit V10 and later.

See Also

“LTransform” (page 1801), “LPoint_Transform_Ex99” (page 1668)
LPoint_Transform_Ex99

LPoint LPoint_Transform_Ex99( LPoint point, LTransform_Ex99 transform );

Description

Applies transform to a point.

Return Values

Values of a new point. The original point is not modified.

Parameters

point Specified point.
transform Specified transformation.

See Also

“LPoint_Transform” (page 1667), “LTransform_Ex99” (page 1802)
Rectangle Functions

“LRect_Set” (page 1670)
“LRect_Transform_Ex99” (page 1672)

Obsolete:
“LRect_Transform” (page 1671)
LRect_Set

\[
\text{LRect} \ LRect\_Set( \ LCoord \ x0, \ LCoord \ y0, \ LCoord \ x1, \ LCoord \ y1 );
\]

Description

Creates an \texttt{LRect} type from the specified lower left and upper right coordinates. A rectangle can be defined by specifying its lower left and the upper right corners.

Return Values

Returns the newly created \texttt{LRect}.

Parameter

\begin{itemize}
    \item \texttt{x0} \hspace{2cm} x- coordinate of the lower left point
    \item \texttt{y0} \hspace{2cm} y- coordinate of the lower left point
    \item \texttt{x1} \hspace{2cm} x- coordinate of the upper right point
    \item \texttt{y1} \hspace{2cm} y- coordinate of the upper right point
\end{itemize}

See Also

“LTransform” (page 1801), “Rectangle Functions” (page 1669)
LRect_Transform

\[ \text{LRect} \] \text{LRect\_Transform}( \text{LRect} \text{ rect}, \text{LTransform} \text{ transform} );

**Description**

Applies \textit{transform} to \textit{rect}.

**Return Values**

Returns a new transformed rectangle. Original \textit{rect} is not modified.

**Parameters**

- \textit{rect} \hspace{1cm} Rectangle that needs to be transformed.
- \textit{transform} \hspace{1cm} Specified transformation.

**Note:**

Note that this function is superseded in L-Edit V10 and later.

**See Also**

“LRect\_Transform\_Ex99” (page 1672), “LTransform” (page 1801),
“Rectangle Functions” (page 1669)
LRect_Transform_Ex99

LRect LRect_Transform_Ex99( LRect rect, LTransform_Ex99 transform );

Description

Applies transform to rect.

Return Values

Returns a new transformed rectangle. Original rect is not modified.

Parameters

  rect           Rectangle that needs to be transformed.
  transform      Specified transformation.

See Also

“Rectangle Functions” (page 1669), “LTransform_Ex99” (page 1802)
Transformation Functions

“LTransform_Set_Ex99” (page 1675)  
“LTransform_Add_Ex99” (page 1679)  
“LTransform_GetInverse” (page 1682)  

Obsolete:  
“LTransform_Set” (page 1674)  
“LTransform_Add” (page 1678)  

“LTransform_Zero_Ex99” (page 1677)  
“LTransform_Subtract_Ex99” (page 1681)  
“LTransform_Zero” (page 1676)  
“LTransform_Subtract” (page 1680)
LTransform_Set

LTransform LTransform_Set( LCoord xtrans, LCoord ytrans, LOrientation orient, LMagnification mag );

Description

Sets a transformation structure.

Return Values

An LTransform structure containing the specified transformation.

Parameters

- **xtrans**: Translation amount in the x-direction.
- **ytrans**: Translation amount in the y-direction.
- **orient**: Orientation.
- **mag**: Magnification.

Note: Note that this function is obsolete in L-Edit V10 and later.

See Also

LTransform_Set_Ex99

LTransform_Set_Ex99( LCoord xtrans, LCoord ytrans, LOrientation_Ex99 orient, LMagnification mag );

Description

Sets a transformation structure.

Return Values

An LTransform_Ex99 structure containing the specified transformation.

Parameters

- **xtrans**: Translation amount in the x-direction.
- **ytrans**: Translation amount in the y-direction.
- **orient**: Orientation as a real number.
- **mag**: Magnification.

See Also

LTransform_Zero

LTransform LTransform_Zero( void );

Description

Makes an identity transformation.

Return Values

Returns the identity transformation.

Note:

Note that this function is obsolete in L-Edit V10 and later.

See Also

“LTransform” (page 1801), “Transformation Functions” (page 1673)
LTransform_Zero_Ex99

LTransform_Zero_Ex99( void );

Description

Makes an identity transformation.

Return Values

Returns the identity transformation.

See Also

“LTransform_Ex99” (page 1802), “Transformation Functions” (page 1673)
LTransform_Add

LTransform LTransform_Add(LTransform transform_to_be_added, LTransform current_transform);

Description

Adds two transformations. A transform is the translation, orientation, or magnification of an object.

Return Values

Returns the sum of transform_to_be_added and current_transform as an LTransform.

Parameters

transform_to_be_added Transformation structure 1.
current_transform Transformation structure 2.

Note: Note that this function is obsolete in L-Edit V10 and later.

See Also

“LTransform” (page 1801), “Transformation Functions” (page 1673)
**LTransform_Add_Ex99**

```c
LTransform_Ex99 LTransform_Add_Ex99( LTransform_Ex99 transform_to_be_added,
                                   LTransform_Ex99 current_transform );
```

**Description**

Adds two transformations. A transform is the translation, orientation, or magnification of an object.

**Return Values**

Returns the sum of `transform_to_be_added` and `current_transform` as an `LTransform`.

**Parameters**

- `transform_to_be_added` Transformation structure 1.
- `current_transform` Transformation structure 2.

**See Also**

“LTransform_Ex99” (page 1802), “Transformation Functions” (page 1673)
LTransform_Subtract

```c
LTransform LTransform_Subtract( LTransform transform_to_be_subtracted,
                                LTransform current_transform );
```

Description

Subtracts `transform_to_be_subtracted` from `current_transform`. A transform is the translation, orientation, or magnification of an object.

Return Values

The resulting `transform` if successful; zero `transform` otherwise.

Parameters

- `transform_to_be_subtracted`: Transformation structure 1.
- `current_transform`: Transformation structure 2.

Note: Note that this function is obsolete in L-Edit V10 and later.

See Also

“LTransform” (page 1801), “Transformation Functions” (page 1673)
**LTransform_Subtract_Ex99**

```c
LTransform_Ex99 LTransform_Subtract_Ex99(
    LTransform_Ex99 transform_to_be_subtracted,
    LTransform_Ex99 current_transform);
```

**Description**

Subtracts one transform from another. A transform is the translation, orientation, or magnification of an object.

**Return Values**

The resulting `transform` if successful; zero `transform` otherwise.

**Parameters**

- `transform_to_be_subtracted` Transformation structure 1.
- `current_transform` Transformation structure 2.

**See Also**

“LTransform_Subtract” (page 1680), “LTransform_Ex99” (page 1802),
“Transformation Functions” (page 1673).
**LTransform_GetInverse**

```c
LTransform_Ex99 LTransform_GetInverse( transform_to_be_inverted );
```

**Description**

Calculates the inverse transform of the given transform.

**Parameters**

- `transform_to_be_inverted` Previous edit transformation.

**Return Values**

Returns the transformation inversion.

**Example**

```c
LTransform_Ex99 oEditTransform;
LWindow_GetEditTransform(pCurrentWindow, &oEditTransform);
oEditTransform = LTransform_GetInverse(oEditTransform);
LPoint ptTranslatedCursorLoc = LPoint_Transform_Ex99(ptCursorLoc,
    oEditTransform);
```

**See Also**

“**LTransform_Ex99**” (page 1802), “**LWindow_GetEditTransform**”, “**LPoint_Transform_Ex99**” (page 1668), “**Transformation Functions**” (page 1673)
Cross Section Functions

“LCSV_Run” (page 1684)
LCSV_Run

```c
void LCSV_Run( LCell incell );
```

Description

Invokes the Generate Cross Section dialog, retrieves its default values, waits for user input and then runs a Cross-Section view for the given cell (`incell`).

Parameters

- `incell`: The cell for which to generate the cross section view.

Examples

```
LCSV_Run(pCell);
```

Version

Available in L-Edit 8.4 and later versions.
T-Cell Functions

LCell_GetParameterAsBoolean (page 1686)
LCell_GetParameterAsCoord (page 1688)
LCell_GetParameterAsCoordOnGrid (page 1688)

LCell_GetParameterAsDouble (page 1689)
LCell_GetParameterAsInt (page 1690)
LCell_GetParameterAsLayer (page 1691)
LCell_GetParameterAsBoolean

LBoolean LCell_GetParameterAsBoolean(LCell pCell, const char * lpszParamName)

Description

Parameters

Examples

Version

Available in L-Edit 15.10 and later versions.
LCell_GetParameterAsCoord

LCoord LCell_GetParameterAsCoord(LCell pCell, const char * lpszParamName, double scale)

Description

Parameters

Examples

Version

Available in L-Edit 15.10 and later versions.
LCell_GetParameterAsCoordOnGrid

LCoord LCell_GetParameterAsCoordOnGrid(LCell pCell, const char * lpszParamName, double scale)

Description

Parameters

Examples

Version

Available in L-Edit 15.10 and later versions.
LCell_GetParameterAsDouble

    double LCell_GetParameterAsDouble(LCell pCell, const char * lpszParamName)

Description

Parameters

Examples

Version

Available in L-Edit 15.10 and later versions.
LCell_GetParameterAsInt

int LCell_GetParameterAsInt(LCell pCell, const char * lpszParamName)

Description

Parameters

Examples

Version

Available in L-Edit 15.10 and later versions.
**LCell_GetParameterAsLayer**

Layer LCell_GetParameterAsLayer(LCell pCell, const char * lpszParamName)

**Description**

**Parameters**

**Examples**

**Version**

Available in L-Edit 15.10 and later versions.
LAtоФ

```
double LAtоФ( const char* szString );
```

**Description**

Function LAtоФ is used whenever a T-Cell is generated and a parameter type is float. It appears to convert a constant character string to a double and handles engineering prefixes (i.e. 5u -> 5E-6).
Data Types and Typedefs

LAmbiguousFillType (page 1695)
LBoolean (page 1697)
LCapType (page 1699)
LCell_GetParameterAsString (page 1701)
LCell_GetParameterName (page 1702)
LArcDirection (page 1696)
LBooleanOperation (page 1698)
LCell (page 1700)
LCIFParam (page 1704)
LCoord (page 1707)
LCursorType (page 1709)
LDerivedLayerAreaOperation (page 1711)
LDerivedLayerDensityOperation (page 1715)
LDerivedLayerParamEx830 (page 1717)
LDesignRuleFlags (page 1721)
LDialogItem (page 1723)
LDrcFlags (page 1725)
LDrcRuleType (page 1727)
LEntity (page 1729)
LEnvironment (page 1730)
LEnvironment (page 1730)
LEXtractOptions (page 1731)
LFile (page 1736)
LGDSParam (page 1738)
LGeomType (page 1746)
LGrid_v10_00 (page 1750)
LINstance (page 1752)
LLLabel (page 1754)
LLayerParamEx830 (page 1757)
LLen (page 1763)
LMarker (page 1765)
LOrientation (page 1770)
LOutlineStyle (page 1772)
LPass (page 1775)
LPassParam (page 1777)
LPieParams (page 1779)
LPort (page 1781)
LColor (page 1706)
LCore (page 1708)
LCurve (page 1710)
LDerivedLayerBoolOperation (page 1713)
LDerivedLayerParam (page 1716)
LDerivedLayerSelectOperation (page 1719)
LDialogInfo (page 1724)
LDrcRule (page 1726)
LDrcStatus (page 1728)
LFile (page 1736)
LExtractOptionsEx840 (page 1733)
LFileType (page 1737)
LDAP (page 1738)
LGeomType (page 1746)
LGridEx840 (page 1748)
LJoinType (page 1753)
LLayerParam (page 1756)
LLayerViewStatus (page 1762)
LMagnification (page 1764)
LObject (page 1769)
LOrientation_Ex99 (page 1771)
LPalette (page 1774)
LPassMode (page 1776)
LPassType (page 1778)
LPoint (page 1780)
LPropertyType (page 1782)
LRect (page 1783)
LRenderingAttribute (page 1784)
LRenderingAttributeIndex (page 1786)
LSelection (page 1788)
LSelectionParam (page 1789)
LShapeType (page 1790)
LSpecialLayer (page 1791)
LStatus (page 1792)
LStipple (page 1793)
technology (page 1795)
LTechnology (page 1797)
LTechnologyEx840 (page 1798)
LTorusParams (page 1800)
LTransform (page 1801)
LTransform_Ex99 (page 1802)
LVertex (page 1803)
LWindow (page 1804)
LWindowType (page 1805)
LWireConfig (page 1806)
LWireConfigBits (page 1807)
LWireParam (page 1808)
UPIDrawingToolType (page 1809)
LAmbiguousFillType

typedef enum
{
    LDo_Not_Flag = 0,
    LFlag,
    LFix
} LAmbiguousFillType;

Description

LDo_Not_Flag   Ignore polygons with ambiguous fills
LFlag          Flag polygons with ambiguous fills
LFix           Fix polygons with ambiguous fills

Note: Note that this data type is superseded in L-Edit V10 and later.

See Also

“LDesignRuleFlags” (page 1721), “LFile_GetDesignRuleFlags” (page 1242),
“LFile_SetDesignRuleFlags” (page 1243)
**LArcDirection**

```c
typedef enum {
    CW,
    CCW
} LArcDirection;
```

**Definition**

Specifies the direction of an arc with respect to its starting vertex. Choose **CW** to specify a clockwise direction or **CCW** to specify a counterclockwise direction.

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“LVertex_AddCurve” (page 1404), “LVertex_GetCurve” (page 1405), “LVertex_SetCurve” (page 1409).
LBoolean

typedef enum {LFALSE, LTRUE} LBoolean;

Description

An enumerated datatype indicating the Boolean value of an L-Edit property value.
**LBooleanOperation**

```c
typedef enum
{
    LBoolOp_OR,
    LBoolOp_AND,
    LBoolOp_XOR,
    LBoolOp_NOT,
    LBoolOp_GROW,
    LBoolOp_SHRINK,
    LBoolOp_SUBTRACT
} LBooleanOperation;
```

**Description**

This data type is used to control the boolean operation engine.

**Version**

This data type is available in V10 and later.

**See Also**

“LCell_BooleanOperation” (page 1318)
LCapType

typedef enum {
    LCapButt, 
    LCapRound, 
    LCapExtend
} LCapType;

Description

Defines the end style of a wire.

See Also

“LWire_GetCapType” (page 1435), “LWire_SetCapType” (page 1443)
LCell

typedef struct _LCell *LCell;

Description

A pointer to an L-Edit cell whose contents can only be accessed or modified through UPI functions.

See Also

“Cell Functions” (page 1260)
**LCell_GetParameterAsString**

```c
const char * LCell_GetParameterAsString(LCell pCell, const char * lpszParamName)
```
LCell_GetParameterName

LTableCellParameterType LCell_GetParameterName(LCell pCell, const char * lpszParamName);
**LCell_GetParameterType**

```c
const char *LCell_GetParameterType( LCell pCell, const char* szPreceedingParameter );
```
**LCIFParam**

```
typedef struct {
    int poly_to_rect;
    int port_rect;
} LCIFParam;
```

**Description**

LCIFParam is used to get and set the CIF setup of a file. Rectangular polygons are read as boxes if `poly_to_rect` is 1. Port boxes are written out if `port_rect` is 1.

**See Also**

“Importing Files” (page 133)
**LOverwriteCellsScopeOnImport**

```c
typedef enum
{
    cOverwriteAllCells = 0,
    cOverwriteCellsInTheTopDesignOnly,
    cDontOverwriteCells
} LOverwriteCellsScopeOnImport;
```

**Description**

LOverwriteCellsScopeOnImport is an enumerated datatype used to specify cell overwrite scope on import. It uses the same options as in GDS or CIF Import dialogs.

**See Also**

"Importing Files" (page 133)
**LColor**

```c
typedef struct {
    short LRed;
    short LBlue;
    short LGreen;
} LColor;
```

**Description**

Defines a color to be used by L-Edit.

**See Also**

“`LFile_GetColorPalette`” (page 1602), “`LFile_SetColorPalette`” (page 1605)
LCoord

typedef long LCoord;

Description

The basic internal unit coordinate type for the L-Edit layout space.

See Also

“Point Functions” (page 1663)
typedef struct _LCore *LCore;

**Description**

A pointer to an L-Edit standard cell place-and-route core, whose contents can only be accessed or modified through UPI functions. A core is generated by the standard cell place and route utility.

**See Also**

“Core Functions” (page 1656)
LCursorType

typedef enum {
    LSnapping,
    LSmooth
} LCursorType;

Description

Lists the cursor’s (mouse pointer’s) modes of movement: bound to the mouse snap grid points ("snapping") or unconstrained ("smooth").

See Also

“LGrid_v10_00” (page 1750),
**LCurve**

typedef struct
{
    int max_segment_per_curve;
    long max_length_of_segment;
    int display_as_approx;
} LCurve;

**Description**

**Members**

- **max_segment_per_curve**: The maximum number of line segments that will be used to replace a curve.
- **max_length_of_segment**: The maximum length of any line segment used in replacing a curve.
- **display_as_approx**: (Currently this is always be set to true, so that curves will be displayed, saved and exported as a series of segments rather than as smooth curves.)

**Version**

Available in L-Edit 8.4 and later versions.

**Note:** Note that this function is obsolete in L-Edit V10 and later.

**See Also**

“LPolygon_StraightenAllCurves” (page 1451), “LFile_GetGridEx840” (page 1224), “LFile_SetCurveSetup” (page 1232)
LDerivedLayerAreaOperation

typedef enum
{
   LDOAT_Range,
   LDOAT_EQ;
} LAreaCheckType;

typedef enum
{
   LDOUT_LocatorUnits,
   LDOUT_TechnologyUnits,
} LAreaUnitType;

typedef struct _LDerivedLayerAreaOperation
{
   char *layer1;
   int not_flag;
   double n1;
   double n2;
   double area;
   LAreaCheckType area_check_type;
   LAreaUnitType area_unit_type;
} LDerivedLayerAreaOperation;

Description

Used to get and set the parameters of a layer generated using area operations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>Name of the existing layer from which the new layer will be created (derived).</td>
<td>The source layer must precede the derived layer in the layer list.</td>
</tr>
<tr>
<td>not_flag</td>
<td>When set, the NOT of the relation is used, so polygons with area outside the specified range are flagged.</td>
<td>0 = area operation 1 = NOT area operation</td>
</tr>
<tr>
<td>n1, n2</td>
<td>Minimum and maximum values (exclusive) of the area range. Valid only when area_check_type is equal to LDOAT_Range.</td>
<td>n1, n2 ≥ 0</td>
</tr>
<tr>
<td>area</td>
<td>Exact area of selected polygons when area_check_type is equal to LDOAT_EQ.</td>
<td>area ≥ 0</td>
</tr>
<tr>
<td>area_check_type</td>
<td>Indicates whether the derived layer area operation is range or equality.</td>
<td>Possible values are: LDOAT_Range LDOAT_EQ</td>
</tr>
<tr>
<td>area_unit_type</td>
<td>Indicates whether the derived layer area is using locator units or current technology units.</td>
<td>Possible values are: LDOUT_LocatorUnits LDOUT_TechnologyUnits</td>
</tr>
</tbody>
</table>
See Also

“LDerivedLayerParam” (page 1716), “LDerivedLayerBoolOperation” (page 1713),
“LDerivedLayerDensityOperation” (page 1715), “LDerivedLayerSelectOperation” (page 1719)
typedef struct _LDerivedLayerBoolOperation
{
    char *src_layer1;
    char *src_layer2;
    char *src_layer3;
    int layer1_not_op;
    long layer1_grow_amount;
    int layer2_not_op;
    long layer2_grow_amount;
    int layer3_not_op;
    long layer3_grow_amount;
    int layer1_bool_layer2;
    int layer2_bool_layer3;
} LDerivedLayerBoolOperation;

typedef enum
{
    LDOST_Inside=0,
    LDOST_Outside,
    LDOST_Hole,
    LDOST_Cut,
    LDOST_Touch,
    LDOST_Enclose,
    LDOST_Overlap,
    LDOST_Vertex,
    LDOST_Density
} LSelectOperationRelationType;

Description

Used to get and set the parameters of a Boolean generated layer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_layer1, src_layer2, src_layer3</td>
<td>Names of existing layers from which the new layer will be created (derived).</td>
<td>All source layer names must precede the derived layer in the layer list.</td>
</tr>
<tr>
<td>layer1_not_op, layer2_not_op, layer3_not_op</td>
<td>When set, the complement of the indicated source layer is used.</td>
<td>0 = source layer 1 = complement of source layer (NOT)</td>
</tr>
<tr>
<td>layer1_grow_amount, layer2_grow_amount, layer3_grow_amount</td>
<td>The amount, in locator units, by which objects on the indicated source layer are grown or shrunk on the derived layer.</td>
<td>Integer values between -WORLD_MAX and +WORLD_MAX, inclusive.</td>
</tr>
<tr>
<td>layer1_bool_layer2, layer2_bool_layer3</td>
<td>Defines the boolean operation (AND or OR) applied to the indicated pair of source layers.</td>
<td>1 = AND 0 = OR</td>
</tr>
</tbody>
</table>
See Also

“LDerivedLayerParamEx830” (page 1717), “LDerivedLayerAreaOperation” (page 1711),
“LDerivedLayerDensityOperation” (page 1715), “LDerivedLayerSelectOperation” (page 1719)
**LDerivedLayerDensityOperation**

```c
typedef struct _LDerivedLayerDensityOperation
{
    char *layer1;
    char *layer2;
    int not_flag;
    double d1;
    double d2;
} LDerivedLayerDensityOperation;
```

```c
typedef enum
{
    LDOAT_Range=0,
    LDOAT_EQ=1
} LAreaCheckType;
```

```c
typedef enum
{
    LDOUT_LocatorUnits=0,
    LDOUT_TechnologyUnits=1
} LAreaUnitType;
```

**Description**

Used to get and set the parameters of a density generated layer. Parameters include:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>layer1, layer2</code></td>
<td>Names of existing layers from which the new layer will be created (derived). The density of <code>layer2</code> is checked with respect to <code>layer1</code>.</td>
<td>All source layer names must precede the derived layer in the layer list.</td>
</tr>
<tr>
<td><code>not_flag</code></td>
<td>When set, indicates a <strong>not density</strong> operation in which the NOT of the density relation is applied.</td>
<td>0 = density operation, 1 = NOT density operation</td>
</tr>
<tr>
<td><code>d1, d2</code></td>
<td>Minimum and maximum values (exclusive) of the density range.</td>
<td>Between 0 and 100 percent (inclusive), where ( d1 \leq d2 ).</td>
</tr>
</tbody>
</table>

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

- “LLayer_GetDerivedParametersEx830” (page 1565), “LLayer_SetDerivedParametersEx830” (page 1567)
LDerivedLayerParam

typedef struct _LDerivedLayerParam {
    int enable_evaluation; /*if 0 evaluation disabled else enabled*/
    char *name; /*Name of the derived layer*/
    char *src_layer1; /*Name of the first source layer*/
    char *src_layer2; /*Name of the second source layer*/
    char *src_layer3; /*Name of the third source layer*/
    int layer1_not_op; /*If NOT operator enabled for 1st source layer*/
    long layer1_grow_amount; /*grow amount for first source layer*/
    int layer2_not_op; /*If NOT operator enabled for 2nd source layer*/
    long layer2_grow_amount; /*grow amount for second source layer*/
    int layer3_not_op; /*If NOT operator is enabled for layer 3*/
    long layer3_grow_amount; /*grow amount for third source layer*/
    int layer1_bool_layer2; /*1=> AND, 0=> OR of 1st &2nd source layer*/
    int layer2_bool_layer3; /*1=> AND, 0=> OR of 1st &3rd source layer*/
} LDerivedLayerParam;

Description

Used to get and set the parameters of a generated layer.

Note: Note that this function is superseded by “LDerivedLayerParamEx830” (page 1717).
**LDerivedLayerParamEx830**

typedef enum {
    LDOT_Bool,
    LDOT_Area,
    LDOT_Select,
    LDOT_Density
} LDerivationType;

typedef enum {
    UNKNOWN = 0,
    CELL_BROWSER,
    TEXT,
    LAYOUT,
    CROSS_SECTION,
    CODE,
    LW_SPICE,
    LW_LOG,
    LW_COMMAND,
    LW_HTML
} LWindowType;

typedef enum {
    L_unassigned=0,
    L_none,
    L_int,
    L_real,
    L_bool,
    L_string,
    L_enum,
    L_byte,
    L_ptr,
    L_blob
} LPropertyType;

typedef enum {
    LTC_notexist=-1,
    LTC_bool,
    LTC_int,
    LTC_float,
    LTC_string,
    LTC_layer,
    LTC_hstretch,
    LTC_vstretch
} LTCellParameterType;

typedef union _LDerivedLayerOperation {
    LDerivedLayerBoolOperation boolean;
    LDerivedLayerSelectOperation select;
    LDerivedLayerAreaOperation area;
    LDerivedLayerDensityOperation density;
} LDerivedLayerOperation;

typedef struct _LDerivedLayerParamEx830 {
    char *name;
    int enable_evaluation;
    LDerivationType derivation_type;
    }
LDerivedLayerParamEx830 associates the specified layer with derivation parameters for one of four derived layer types: Boolean, Select, Area, or Density. Its member `derivation_type` of type `LDerivationType` indicates the derivation type of the derived layer, and its member `operation` of type `LDerivedLayerOperation` points to the parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the layer for which derivation parameters are defined.</td>
<td>Any valid layer name.</td>
</tr>
<tr>
<td>enable_evaluation</td>
<td>Flag to enable derivation on the specified layer. When this flag is not set, the layer is not considered derived.</td>
<td>0 = No derivation. 1 = Derivation enabled.</td>
</tr>
<tr>
<td>derivation_type</td>
<td>Indicates the type of derivation for which parameters are set. There are four possible derivation types: Boolean, Area, Select, and Density.</td>
<td>Possible values are: LDOT_Bool LDOT_Area LDOT_Select LDOT_Density</td>
</tr>
<tr>
<td>operation</td>
<td>Pointer to derivation parameters corresponding to the type specified by <code>derivation_type</code>.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** This function supersedes “LDerivedLayerParam” (page 1716). Also, `LDerivedLayerParamEx00` can be used interchangeably with `LDerivedLayerParamEx830`.

**Version**

Available in L-Edit 8.3 and later versions.

**See Also**

**LDerivedLayerSelectOperation**

```c
typedef enum {
    LDOST_Inside,
    LDOST_Outside,
    LDOST_Hole,
    LDOST_Cut,
    LDOST_Touch,
    LDOST_Enclose,
    LDOST_Overlap,
    LDOST_Vertex,
    LDOST_Density
} LSelectOperationRelationType;

typedef struct _LDerivedLayerSelectOperation {
    char *layer1;
    char *layer2;
    int not_flag;
    int range_enabled_flag;
    int n1;
    int n2;
    double d1;
    double d2;
    LSelectOperationRelationType relation_type;
} LDerivedLayerSelectOperation;
```

**Description**

`LDerivedLayerSelectOperation` is used to get and set the parameters of a layer generated using select operations. Its member `relation_type` of type `LSelectOperationRelationType` specifies the select relationship applied.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>layer1, layer2</code></td>
<td>Existing layers from which the new layer will be created (derived).</td>
<td>Both layer names must precede the derived layer in the layer list.</td>
</tr>
<tr>
<td><code>not_flag</code></td>
<td>When set, indicates a not select operation in which the NOT of the select relation is applied.</td>
<td>0 = select operation, 1 = NOT select operation</td>
</tr>
<tr>
<td><code>range_enabled_flag</code></td>
<td>When set, indicates that the select relationship will be defined by a range of vertices. This flag is valid only if <code>relation_type</code> is equal to <code>LDOST_Vertex</code>.</td>
<td>0 = range disabled, 1 = range enabled</td>
</tr>
<tr>
<td><code>n1, n2</code></td>
<td>The minimum and maximum values (exclusive) of the vertex count used for vertex selection operations. Valid only if <code>range_enabled_flag</code> is equal to 1 and <code>relation_type</code> is equal to <code>LDOST_Vertex</code>.</td>
<td>Integers greater than or equal to 0, where <code>n1 ≤ n2</code>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Definition</td>
<td>Values</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>d1, d2</td>
<td>The minimum and maximum values (exclusive) of the density range used for density select operations. Valid only if relation_type is equal to LDOST_Density.</td>
<td>Between 0 and 100 percent (inclusive), where $d1 \leq d2$.</td>
</tr>
<tr>
<td>relation_type</td>
<td>Specifies the type of select relationship used for derivation.</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDOST_Inside</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDOST_Outside</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDOST_Hole</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDOST_Cut</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDOST_Touch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDOST_Enclose</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDOST_Overlap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDOST_Vertext</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LDOST_Density</td>
</tr>
</tbody>
</table>

**Note:** Note that this data type is superseded in L-Edit V10 and later.

**See Also**

LDesignRuleFlags

typedef struct
{
    int FlagSelfIntersection,
    LAmbiguousFillType PolygonsWithAmbiguousFills,
    int FlagIgnoredObjects
    int FlagOffGridObjects
    double GridSize
    int UseLocatorUnits
} LDesignRuleFlags;

Description

FlagSelfIntersection    Ignore polygons with ambiguous fills.
PolygonsWithAmbiguousFills    Flag polygons with ambiguous fills, fix polygons with ambiguous fills, or ignore them.
FlagIgnoredObjects    Flag objects not checked by DRC.
FlagOffGridObjects    Flags off-grid objects.
GridSize    Used to flag off-grid objects.
UseLocatorUnits    Indicates whether GridSize is in technology units or locator units.

See Also

“LAmbiguousFillType” (page 1695), “LFile_GetDesignRuleFlags” (page 1242),
“LFile_SetDesignRuleFlags” (page 1243)
LDesignRuleParam

typedef struct LDesignRuleParam {
    int enable; /*0=>disabled, 1=>enabled*/
    char *name; /*Name of the design rule*/
    LDrCRuleType rule_type; /*type of a design rule*/
    int ignore_coincidences; /*0=>false, 1=>true*/
    int ignore_intersections; /*0=> false, 1=>true*/
    int ignore_enclosures; /*0=> false, 1=>true*/
    int ignore_45_acute_angles; /*0=> false, 1=>true*/
    char *layer1; /*Name of the first layer involved in design rule*/
    char *layer2; /*Name of the second layer involved in design rule*/
    long distance; /*Distance value associated with a rule*/
    int use_internal_units; /*0=> false, 1=>true :False=> use LAMBDA*/
} LDesignRuleParam;

typedef enum
{
    LDOT_Bool=0,
    LDOT_Area=1,
    LDOT_Select=2,
    LDOT_Density=3,
    LDOT_External=4
} LDerivationType;

Description

This structure is used to get and set parameters of a design rule.

See Also

“LDrcRule_GetParameters” (page 1630), “LDrcRule_SetParameters” (page 1631),
“LDrcRule_Add” (page 1623)
LDialogItem

typedef struct {
    char prompt[40];
    char value[21];
} LDialogItem;

Description

Defines the prompt and value fields associated with a multiple-line input dialog. This structure is used by LDialog_MultiLineInputBox.

See Also

“Dialog Functions” (page 1076)
LDisplayUnitInfo

typedef struct
{
    char     szDispUnitName[128];
    double   dScaleFactor;
    int      nEditDecimalDigits;
    int      nMouseGridDecimalDigits;
} LDisplayUnitInfo;

Description

This struct is used to control the display of L-Edit internal units in the locator bar.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szDispUnitName</td>
<td>Name of display unit</td>
</tr>
<tr>
<td>dScaleFactor</td>
<td>Internal units divided by dScaleFactor equals Display units.</td>
</tr>
<tr>
<td>nEditDecimalDigits</td>
<td>Number of decimal digits to use when formatting string for editing.</td>
</tr>
<tr>
<td>nMouseGridDecimalDigits</td>
<td>Number of decimal digits to use when formatting string for display of mouse position.</td>
</tr>
</tbody>
</table>

Version

This data type is available in V10 and later.

See Also

“LFile_GetDisplayUnitInfo” (page 1249)
**LDrcFlags**

```c
typedef struct {
    LBoolean bFlagAcuteAngles;
    LBoolean bFlagAllAngleEdges;
    LBoolean bFlagOffGridObjects;
} LDrcFlags;
```

**Description**

This data type is used to control the DRC tests for acute angle polygons, all-angle edges, and off-grid vertices.

**Version**

This data type is available in V10 and later.

**See Also**

"LFile_GetDrcFlags" (page 1637), "LFile_SetDrcFlags" (page 1638)
LDrcRule

typedef struct _LDrcRule *LDrcRule;

Description

A pointer to an L-Edit design rule whose contents can only be accessed or modified through UPI functions.

See Also

“LDrcRule_Find” (page 1625), “LDrcRule_GetList” (page 1626)
LDrcRuleType

```c
typedef enum { 
    LMIN_WIDTH, 
    LEXACT_WIDTH, 
    LOVERLAP, 
    LEXTENSION, 
    LNOT_EXISTS, 
    LSPACING, 
    LSURROUND, 
    LDENSITY
} LDrcRuleType;
```

**Description**

LDrcRuleType is an enumerated datatype used to specify the type of a design rule.

**See Also**

“LDesignRuleParam” (page 1722)
**LDrcStatus**

```c
typedef enum {
    LDrcStatus_Needed,
    LDrcStatus_Passed,
    LDrcStatus_Failed
} LDrcStatus;
```

**Description**

This data type is used to report the DRC status of a cell.

**Version**

This data type is available in V10 and later.

**See Also**

"[LCell_GetDRCStatus](page 1642)"
**LEntity**

```c
typedef struct _LEntity *LEntity;
```

**Description**

A pointer to an L-Edit entity whose contents can only be accessed or modified through UPI functions.

**Version**

Available in L-Edit 8.4 and later versions.

**See Also**

“Entity Functions” (page 1345).


LEnvironment

typedef struct _LEnvironment {
  short MenuBackgroundColor;
  short MenuForegroundColor;
  short MenuSelectColor;
  short AlertBackgroundColor;
  long DefaultPortTextSize;
  int DropDownMenus;
  int ActivePushRubberbanding;
  int AutoPanning;
  int StatusBar;
  int HideInsides;
  short HorizontalPixels;
  short VerticalPixels;
} LEnvironment;

Description

Used to get and set the environment of a design file. All colors take values between 0 and 15. The int quantities take values of either 0 or 1, equivalent to the off and on states of the corresponding switches in the Setup Application dialog.

Note: The color parameters are not applicable to L-Edit for Windows.
LExtractOptions

typedef struct _LExtractOptions
{
    char szExtDefnFile[256];
    char szExtOutFile[256];
    double dExtractBinSize;
    int iWriteNodeNames;
    int iWriteDeviceCoord;
    int iWriteShortedDevices;
    int iWriteParasiticCap;
    double dParasiticCutoff;
    int iWriteNodesAs;
    int iWriteSciNotation;
    int iWriteVerboseSPICE;
    char *szExtIncludeStmt;
    int iLabelAllDevices;
    LLayer oDeviceLabelLayer;
    int iSubCktRecognition;
    LLayer oSubCktRecogLayer;
    int iUseSubCktNetlistFmt;
    int iFlagImproperOverlaps;
    LLayer oIgnoreConnPortLayer;
    char szIgnoreConnPort[256];
    char szIgnoreCrossPort[256];
    long lMaxIncludeStmtLen;
} LExtractOptions;

Description

Used to get and set the extract options for a cell. The int quantities take values of either 0 or 1, equivalent to the off and on states of the corresponding switches in the Extract dialog. All options available in the extract dialog can be set with the above structure.

Note:

Note that this data type is superseded in L-Edit V10 and later.
General Options

- **szExtDefnFile**: Character string of the extract definition file. (256 characters max).
- **szExtOutFile**: Character string of the extract SPICE output file. (256 characters max).
- **dExtractBinSize**: Bin size in locator units.

Output Options

- **iWriteNodeNames**: Write node names in comments. (0 - False, Otherwise True).
- **iWriteDeviceCoord**: Write device coordinates in comments. (0 - False, Otherwise True).
- **iWriteShortedDevices**: Write shorted devices in comments. (0 - False, Otherwise True).
- **iWriteParasiticCap**: Write parasitic capacitances. (0 - False, Otherwise True).
- **dParasiticCutoff**: Cutoff value for parasitic capacitors. (in Femtofarads).
- **iWriteNodesAs**: Write nodes as (integers or names). (0 - Integers, Otherwise Names).
- **iWriteSciNotation**: Write values in scientific notation. (0 - False, Otherwise True).
- **iWriteVerboseSPICE**: Write R, L, C with verbose style (R=, L=, C=). (0 - False, Otherwise True).
- **szExtIncludeStmt**: SPICE include statement.
- **iLabelAllDevices**: Create ports for all devices. (0 - False, Otherwise True).
- **oDeviceLabelLayer**: Place device labels on this layer.

Subcircuit Options

- **iSubCktRecognition**: Recognize subcircuit instances. (0 - False, Otherwise True).
- **oSubCktRecogLayer**: Subcircuit recognition layer.
- **iUseSubCktNetlistFmt**: Write netlist as a subcircuit. (0 - False, Otherwise True).
- **iFlagImproperOverlaps**: Flag improper overlaps. (0 - False, Otherwise True).
- **oIgnoreConnPortLayer**: Ignore connection ports on this layer.
- **szIgnoreConnPort**: Ignore connection ports with this name.
- **szIgnoreCrossPort**: Ignore cross ports with this name.

Miscellaneous

- **lMaxIncludeStmtLen**: Length of the .include statement string.
typedef struct _LExtractOptionsEx840
{
    // General Options
    char szExtDefnFile[256];
    char szExtOutFile[256];
    double dExtractBinSize;
    LBoolean bLabelAllDevices;
    char szDeviceLabelLayer[256];

    // Output Options
    LBoolean bWriteNodeNames;
    LBoolean bWriteDeviceCoord;
    LBoolean bWriteShortedDevices;
    LBoolean bWriteCapResWarnings;
    LBoolean bWriteParasiticCap;
    double dParasiticCutoff;
    LBoolean bWriteNodesAsNames;
    LBoolean bWriteSciNotation;
    LBoolean bWriteVerboseSPICE;
    LBoolean bWriteSubCktDefs;
    LBoolean bWriteENDStatement;
    char* pszExtIncludeStmt;

    // Subcircuit Options
    LBoolean bSuCktRecognition;
    char szSubCktRecogLayer[256];
    LBoolean bUseSubCktNetlistFmt;
    LBoolean bFlagImproperOverlaps;
    char szIgnoreConnPortLayer[256];
    char szIgnoreConnPort[256];
    char szIgnoreCrossPort[256];

    // Miscellaneous
    long nMaxIncludeStmtLen;
} LExtractOptionsEx840

Description

Used to get and set the Extract options for a cell. All options available in the Extract dialog can be set with the above structure.

Note: In order to properly retrieve the .include statement, the data member pszExtIncludeStmt of this structure must be dynamically allocated to a sufficient size. The data member nMaxIncludeStmtLen must be set to the size of the allocated string pszExtIncludeStmt. Failure to do so could result in a general protection fault. See sample code in the descriptions of “LExtract_RunEx840” (page 1649), “LExtract_GetOptionsEx840” (page 1652), and “LExtract_SetOptionsEx840” (page 1654).
Chapter 34: UPI Functions Reference

General Options

- **szExtDefnFile**: Character string of the extract definition file. (256 characters max).
- **szExtOutFile**: Character string of the extract SPICE output file. (256 characters max).
- **dExtractBinSize**: Bin size in locator units.
- **szDeviceLabelLayer**: Character string identifying the layer on which to place device labels (256 characters max). Leave blank for the recognition layer.

Output Options

- **bWriteNodeNames**: Write node names in comments.
- **bWriteDeviceCoord**: Write device coordinates in comments.
- **bWriteShortedDevices**: Write shorted devices in comments.
- **bWriteCapResWarnings**: Write layer capacitance and resistance warnings.
- **bWriteParasiticCap**: Write parasitic capacitances.
- **dParasiticCutoff**: Cutoff value for parasitic capacitors (in Femtofarads).
- **bWriteNodesAsNames**: Write nodes as names. When false, writes nodes as integers.
- **bWriteSciNotation**: Write values in scientific notation.
- **bWriteVerboseSPICE**: Write R, L, C with verbose style (R=, L=, C=).
- **bWriteSubCktDefs**: Write empty subcircuit definitions.
- **bWriteENDStatement**: Write the .END statement.
- **szExtIncludeStmt**: Pointer to a SPICE .include statement.

Subcircuit Options

- **bSubCktRecognition**: Recognize subcircuit instances.
- **szSubCktRecogLayer**: Character string identifying the subcircuit recognition layer.
- **bUseSubCktNetlistFmt**: Write netlist as a subcircuit.
- **bFlagImproperOverlaps**: Flag improper overlaps.
- **szIgnoreConnPortLayer**: Ignore connection ports on this layer, specified by a character string.
- **szIgnoreConnPort**: Ignore connection ports with this name, specified by a character string.
- **szIgnoreCrossPort**: Ignore cross ports with this name, specified by a character string.

Miscellaneous

- **nMaxIncludeStmtLen**: Length of the .include statement string.
Version

Available in L-Edit 8.4 and later versions.

See Also

“LExtract_GetOptionsEx840” (page 1652), “LExtract_SetOptionsEx840” (page 1654),
“LExtract_RunEx840” (page 1649)
LFile

typedef struct _LFile *LFile;

Description

A pointer to an L-Edit layout file whose contents can only be accessed or modified through UPI functions.

See Also

“File Functions” (page 1193)
**LFileType**

```c
typedef enum {
    LTdbFile,
    LCifFile,
    LGdsFile,
    LV6TdbFile
} LFileType;
```

**Description**

Lists the design formats supported by L-Edit: Tanner Database (TDB) format, Caltech Intermediate Form (CIF), and GDS II (stream) format. CIF and GDS II are standard machine-readable formats for representing IC layouts. Also, LV6TdbFile is necessary in the special case of opening L-Edit V6 or previous TDB files.

**See Also**

“LFile_Open” (page 1196), “LFile_SaveAs” (page 1199)
### LGDSParam

```c
typedef struct {
    int upcase_cell_name;
    short circle_to_polygon_sides; /* OBSOLETE */
    int use_default_units;
} LGDSParam;
```

#### Description

**Note:** LGDSParam supersedes LFile_SetGDSParameters and LFile_SetGDSParameters for GDS export options.

LGDSParam is used to get and set the GDSII setup of a file. If `upcase_cell_name` is 1, L-Edit will write out all cells with uppercase names. A circle is written out as n-sided polygons with its vertices snapped to the Manufacturing grid set for the design.

A log file can be generated when LFile_SaveAs is used to export in GDS format, if that option is set in the File > GDS Export dialog.

#### Version

In L-Edit version 10 and earlier, the structure for LGDSParam had 3 members, `upcase_cell_name` to write out cells with uppercase names, `circle_to_polygon_sides` to write circles as n-sided polygons and `use_default_units` to use default GDSII units.

In L-Edit version 10 and later, the structure member `circle_to_polygon_sides` is no longer supported. Instead, during GDSII export, a circle is approximated as an all-angle polygon with its vertices snapped to the Manufacturing grid set for the design.

#### See Also

typedef struct
{
    const char* cszDestFileName;
    // Destination file name
    LBoolean bZipOutputFile;
    LGDSExportScope ExportScope;
    // Export scope value enumeration, see below for values.
    char* cszSpecifiedCell;
    // Contains Cell Name if ExportScope is gdsExportSpecifiedCell
    const char* const * cpszIncludeLibraries;
    // Libraries to include, where ExportScope is gdsExportCellsFromLibraries
    LBoolean bIncludeHierarchy;
    // LTRUE if hierarchy of specified cells need to be included in the export, o else LFALSE.
    char* const * cpszExcludeLibraries;
    // List of libraries to exclude (empty or NULL if none)
    LBoolean bUseDefaultUnits;
    // LTRUE if Use default units
    double dMicrons;
    double dUserUnits;
    // Two values that specify custom Units, if bUserDefaultUnits is LFALSE
    int nUpcaseCellName;
    // 0 for PFile::GDS_CasePreserve, 1 for PFile::GDS_CaseUpper,
    // 2 for PFile::GDS_CaseLower
    int nCellNameLength;
    // Restrict cell name character count or 32 - Standard
    const char* cszMapFileName;
    // Do not export hidden objects and layers
    LBoolean bOverwriteGDSIIDataType;
    // Overwrite object data type with layer data type
    LBoolean bCalcChecksum;
    // Calculate MOSIS checksum
    LBoolean bCheckSelfIntersections;
    // Check self-intersecting polygons and wires
    LBoolean bFracture;
    // Fracture polygons with more than nFractureLimit vertices
    int nFractureLimit;
    // Maximum number of vertices
} LGDSPParamEx;

Description

LGDSPParamEx exports GDSII to a given destination file.

In L-Edit version 10 and later, the structure member circle_to_polygon_sides is no longer supported. Instead, during GDSII export, a circles is approximated as an all-angle polygon with its vertices snapped to the manufacturing grid set for the design.
See Also

LGDSExport Scope

typedef enum
{
    gdsExportAllCells,
    gdsExportActiveCell,
    gdsExportFabricationCell,
    gdsExportSpecifiedCell,
    gdsExportCellsFromLibraries,
} LGDSExportScope;

Description

Lists the scope of cells to be exported to GDSII.

Example 1

Exports cells from given libraries, and demonstrates how to specify excluded libraries.

    // Export Cells from Libraries
    const char* cszLibrariesToExportFrom[3]; // 2 libraries. Need one more
    array member to set to NULL to indicate the end
    cszLibrariesToExportFrom[0] = "Mosfet";
    cszLibrariesToExportFrom[1] = "Buffer";
    cszLibrariesToExportFrom[2] = NULL;

    const char* cszExcludeLibraries[2]; // 1 library. Need one more array
    member to set to NULL to indicate the end
    cszExcludeLibraries[0] = "Buffer";
    cszExcludeLibraries[1] = NULL;

    GDSParams.cszDestFileName = "test_ExportGDSII_CellsFromLibraries.gds";
    GDSParams.ExportScope = gdsExportCellsFromLibraries;

    //GDSParams.cszSpecifiedCell = szCellName; // Will not be using this
    member
    GDSParams.cpszIncludeLibraries = cszLibrariesToExportFrom;
    GDSParams.bIncludeHierarchy = LTRUE;
    GDSParams.cpszExcludeLibraries = cszExcludeLibraries; // excluded
    libraries will be exported as links...
    GDSParams.bUseDefaultUnits = LTRUE;
    /* Since bUseDefaultUnits is true the next two parameters won't be
    used this time.
    GDSParams.dMicrons = 0;
    GDSParams.dUserUnits = 0;
    */
    GDSParams.nUpcaseCellName = 0; // 0 - preserve case, 1 - Uppercase, 2
    - Lowercase
    GDSParams.nCellNameLength = 32; // 32 - Standard, 128 - Virtuoso
    GDSParams.cszMapFileName = NULL;

    GDSParams.bDoNotExportHiddenObjects = LFALSE;
    GDSParams.bOverwriteGDSIIDataType = LTRUE;
    GDSParams.bCalcChecksum = LFALSE;
    GDSParams.bCheckSelfIntersections = LTRUE;
Example 2

Exports the Active Cell and its hierarchy.

```c
#include <cstdlib>
#include <cstdarg>
#include <cstdio>
#include <cstring>
#include <ctype>
#include <cmath>
#include <string>
#define EXCLUDE_LEDIT_LEGACY_UPI
#include <ldata.h>
/* Begin -- Uncomment this block if you are using L-Comp. */
/*!include <lcomp.h>
  /* End */
```
extern "C" {
    void Export_Specified_Cell(void);
    void Export_Cells_From_Libraries(void);
    int UPI_Entry_Point(void);
}

void Export_Specified_Cell(void)
{
    LCell pCell = LCell_GetVisible();
    if(!pCell)
    {
        LDialog_MsgBox("No active cell.");
        return;
    }
    LFile pFile = LCell_GetFile(pCell);
    if(!pFile)
    {
        LDialog_MsgBox("Can not get file pointer for active cell.");
        return;
    }
    LGDSParamEx GDSParams;

    /***************************************************************************/
    // Export Specified Cell
    char szCellName[2048];
    LCell_GetName(pCell, szCellName, 2048);
    GDSParams.cszDestFileName = "test_ExportGDSII_SpecifiedCell.gds";
    GDSParams.ExportScope = gdsExportSpecifiedCell;

    GDSParams.cszSpecifiedCell = szCellName;  // current Active cell name
    GDSParams.cpszIncludeLibraries = NULL; // not going to use it since we
    are specifying a cell to export
    GDSParams.bIncludeHierarchy = LTRUE;
    GDSParams.cpszExcludeLibraries = NULL; // excluded libraries will be
    exported as links...
    GDSParams.bUseDefaultUnits = LTRUE;
    /* Since bUseDefaultUnits is true the next two parameters won't be
    used this time.
    GDSParams.dMicrons = 0;
    GDSParams.dUserUnits = 0;*/
    GDSParams.nUpcaseCellName = 0; // 0 - preserve case
    GDSParams.nCellNameLength = 32; // 32 - Standard, 128 - Virtuoso
    GDSParams.cszMapFileName = NULL;

    GDSParams.bDoNotExportHiddenObjects = LFALSE;
    GDSParams.bOverwriteGDSIIDataType = LTRUE;
    GDSParams.bCalcChecksum = LFALSE;
    GDSParams.bCheckSelfIntersections = LTRUE;

    GDSParams.bFracture = LTRUE;
    GDSParams.nFractureLimit = 50;
    GDSParams.bZipOutputFile = LFALSE;

    LStatus status = LFile_ExportGDSII(pFile, &GDSParams, NULL);
if(LStatusOK == status)
    LDialog_MsgBox("Exported Successfully!");
else
    {
        if(LBadFile == status)
            LDialog_MsgBox("Export Failed. Need a valid file pointer to proceed.");
        else if(LBadParameters == status)
            LDialog_MsgBox("Export Failed. No GDS Parameters were supplied or Destination File name is not given.");
        else
            LDialog_MsgBox("Export Failed.");
    }
}

void Export_Cells_From_Libraries(void)
{
    LCell pCell = LCell_GetVisible();
    if(!pCell)
        {
            LDialog_MsgBox("No active cell.");
            return;
        }
    LFile pFile = LCell_GetFile(pCell);
    if(!pFile)
        {
            LDialog_MsgBox("Can not get file pointer for active cell.");
            return;
        }
    LGDSParamEx GDSParams;
    LGDSExportLogParams LogParams;

    See Also

**LGDSExportLogParams**

```c
typedef struct {
    const char* szLogFileName;
    LBoolean bOpenLogInWindow; // if not in QuietMode
} LGDSExportLogParams;
```

**Description**

A log file can be generated when `LFile_SaveAs` is used to export in GDS format, if that option is set in the `File > GDS Export` dialog.

**See Also**

LGeomType

typedef enum {
    LOrthogonal,
    LFortyFive,
    LAllAngle,
    LCurved,
    LNonGeometric,
    LManhattan = LOrthogonal,
    LBoston = LFortyFive,
} LGeomType;

Description

An enumerated datatype indicating the geometry type of an L-Edit object.

See Also

“LObject_GetGeometry” (page 1381)
LGrid

typedef struct {
    long displayed_grid_size;
    long min_grid_pixels;
    long mouse_snap_grid_size;
    LCursorType cursor_type;
    long locator_scaling;
} LGrid;

Description

Used to get and set the grid parameters of the design file. The fields appear as corresponding items in the Setup > Design—Grid.

Note: Note that this function is superseded by “LGridEx840” (page 1748) and “LGrid_v10_00” (page 1750).
LGridEx840

typedef struct {
  long displayed_grid_size;
  long min_grid_pixels;
  long displayed_majorgrid_size;
  long min_majorgrid_pixels;
  long mouse_snap_grid_size;
  LCursorType cursor_type;
  long locator_scaling;
} LGridEx840;

Description

Used to get and set the grid parameters of the design file, which appear as corresponding items in Setup > Design—Grid.

Members

The valid range for numerical values is $1 \leq \text{value} \leq \text{WORLD_MAX}$.

- **displayed_grid_size**: The absolute spacing, in Internal Units, of the minor grid display.
- **min_grid_pixels**: The number of screen pixels per grid square side below which L-Edit hides the minor grid.
- **displayed_majorgrid_size**: The absolute spacing, in Internal Units, of the major grid display.
- **min_majorgrid_pixels**: The number of screen pixels per grid square side below which L-Edit hides the major grid.
- **mouse_snap_grid_size**: Absolute spacing of the mouse snap grid. The value entered in this field is the length, in Internal Units, of a grid square side.
- **cursor_type**: Specifies behavior of the mouse pointer. Possible values are:
  - **LSnapping**: causes the pointer to snap to points on the mouse snap grid.
  - **LSmooth**: allows the pointer to be unconstrained.
- **locator_scaling**: The number of Internal Units equivalent to one Locator Unit.

Examples

```c
/* Get the current grid setting for MyFile */
LGridEx840 Grid;
LFile_GetGridEx840(MyFile, &Grid);

/* Specify new grid settings */
Grid.min_majorgrid_pixels = 10*Grid.min_grid_pixels;
Grid.displayed_majorgrid_size = 10*Grid.displayed_grid_size;

/* Apply the new grid structure to MyFile */
LFile_SetGridEx840(MyFile, &Grid);
```
Version

Available in L-Edit 8.4 and later versions.

**Note:**

Note that this data type is superseded by “LGrid_v10_00” (page 1750).

**See Also**

“LFile_GetGridEx840” (page 1224), “LFile_SetGridEx840” (page 1229), “LCursorType” (page 1709)
LGrid_v10_00

Description

typedef struct
{
  long displayed_grid_size; /* In internal units */
  long min_grid_pixels;
  long displayed_majorgrid_size; /* In internal units */
  long min_majorgrid_pixels;
  long mouse_snap_grid_size; /* In internal units */
  LCursorType cursor_type;
  long locator_scaling;
  long manufacturing_grid_size; /* In internal units */
  LBoolean display_curves_using_manufacturing_grid;
} LGrid_v10_00;

Description

Used to get and set the grid parameters of the design file, which appear as corresponding items in Setup > Design—Grid.

Members

The valid range for numerical values is $1 \leq \text{value} \leq \text{WORLD\_MAX}$.

- **displayed_grid_size**
  The absolute spacing, in Internal Units, of the minor grid display.

- **min_grid_pixels**
  The number of screen pixels per grid square side below which L-Edit hides the minor grid.

- **displayed_majorgrid_size**
  The absolute spacing, in Internal Units, of the major grid display.

- **min_majorgrid_pixels**
  The number of screen pixels per grid square side below which L-Edit hides the major grid.

- **mouse_snap_grid_size**
  Absolute spacing of the mouse snap grid. The value entered in this field is the length, in Internal Units, of a grid square side.

- **cursor_type**
  Specifies behavior of the mouse pointer. Possible values are:
  - **LSnapping**—causes the pointer to snap to points on the mouse snap grid.
  - **LSmooth**—allows the pointer to be unconstrained.

- **locator_scaling**
  The number of Internal Units equivalent to one Locator Unit.

- **manufacturing_grid_size**
  The size of the manufacturing grid, in Internal Units.

- **display_curves_using_manufacturing_grid**
  If true, polygonize all curve for rendering using the manufacturing grid. This previews the effect of streaming these objects to GDSII.
Examples

/* Get the current grid setting for MyFile */
LGrid_v10_00 Grid;
LFile_GetGrid_v10_00( MyFile, &Grid );

/* Specify new grid settings */
Grid.min_majorgrid_pixels = 10 * Grid.min_grid_pixels;
Grid.displayed_majorgrid_size = 10 * Grid.displayed_grid_size;

/* Apply the new grid structure to MyFile */
LFile_SetGrid_v10_00(MyFile, &Grid);

Version

Available in L-Edit 10 and later versions.

See Also

“LFile_GetGrid_v10_00” (page 1225), “LFile_SetGrid_v10_00” (page 1226), “LCursorType” (page 1709)
LInstance

typedef struct _LInstance *LInstance;

Description

A pointer to an L-Edit instance whose contents can only be accessed or modified through UPI functions.

See Also

“Instance Functions” (page 1324)
LJoinType

typedef enum {
    LJoinMiter,
    LJoinRound,
    LJoinBevel,
    LJoinLayout
} LJoinType;

Description

Defines the join style of a wire.

See Also

“LWire_SetJoinType” (page 1442), “LWire_GetJoinType” (page 1436)
LLabel

typedef struct _LLabel *LLabel;

Description

A pointer to an L-Edit label whose contents can only be accessed or modified through UPI functions. A label is a text whose location in a cell is specified by a point. Each cell has a single list of labels. Each label has a layer, text, and a location associated with it.

Version Compatibility

New with version 16.00.

See Also

“LTerminalType” (page 1799), “Port Functions” (page 1452).
LLayer

typedef struct _LLayer *LLayer;

Description

A pointer to an L-Edit layer whose contents can only be accessed or modified through UPI functions.

See Also

“Layer Functions” (page 1528)
LLayerParam

```c
typedef struct {
    char CIFName [7];
    short GDSNumber;
    double cap;
    double rho;
    int lock;
    LLayerViewStatus viewStatus;
    LWireParam wireParam;
} LLayerParam;
```

**Description**

A structure where layer information can be stored. It specifies the CIF name, GDS number, capacitance, and resistance of a layer.

**Return Values**

When lock is zero, a layer is locked.

**Parameters**

- `viewStatus` Indicates whether a layer is visible or hidden.
- `wireParam` Specifies the properties of a wire that can be drawn using this layer.

**Note:**

Note that this data type is superseded in L-Edit V10 and later.

**See Also**

**LLayerParamEx830**

typedef struct {
    char CIFName[7];
    short GDSNumber;
    short GDSDataType;
    double AreaCapacitance;
    double FringeCapacitance;
    double Resistivity;
    LBoolean Locked;
    LBoolean Hidden;
    LWireParam WireParam;
} LLayerParamEx830;

**Description**

`LLayerParamEx830` is an extended layer parameter structure that stores the parameters of a layer such as CIF name, GDSII layer number, GDSII layer data type, area and fringe capacitance, resistivity, default wire parameters, and lock and hidden states.

**Members**

- **cifName**
  CIF layer name. It can be 6 characters or less.

- **GDSNumber**
  GDSII layer number. `-32,768 <= GDSNumber <= 32,767`. The GDSII standard requires `0 <= GDSNumber <= 255`.

- **GDSDataType**
  GDSII data type. `-32,768 <= GDSDataType <= 32,767`. The GDSII standard requires `0 <= GDSNumber <= 255`. A value of `-1` indicates an unassigned GDSII data type.

- **AreaCapacitance**
  Area capacitance in aF/Sq. micron from the layer to the substrate. `0 <= AreaCapacitance` or `AreaCapacitance = -1`. A value of `-1` indicates an unassigned area capacitance.

- **FringeCapacitance**
  Fringe capacitance in fF/micron from the layer to the substrate. `0 <= FringeCapacitance` or `FringeCapacitance = -1`. A value of `-1` indicates an unassigned fringe capacitance.

- **Resistivity**
  Sheet resistance of the layer in ohms/square. `0 <= Resistivity`.

- **Locked**
  Indicates whether the layer is locked.

- **Hidden**
  Indicates whether the layer is hidden.

- **WireParam**
  Default wire parameters of the layer.

**Example**

```c
LFile pFile = LFile_GetVisible();
if(Assigned(pFile))
{
    if(LLayer_New(pFile, LLayer_GetList(pFile), "Metal1") == LStatusOK)
    {
        LLayer pLayer = LLayer_Find(pFile, "Metal1");
        if(Assigned(pLayer))
        {
```

```c
```
LLayerParamEx830 LayerParameters;
strcpy(LayerParameters.CIFName, "CMF");
LayerParameters.GDSNumber = 49;
LayerParameters.GDSDataType = -1;
LayerParameters.AreaCapacitance = 36;
LayerParameters.FringeCapacitance = 0.086;
LayerParameters.Resistivity = 0.08;
LayerParameters.Locked = LFALSE;
LayerParameters.Hidden = LFALSE;
LayerParameters.WireParam.defaultWireWidth =
LFile_LocUtoIntU(pFile, 1.5);
LayerParameters.WireParam.defaultWireMiterAngle = 90;
LayerParameters.WireParam.capType = LCapExtend;
LayerParameters.WireParam.joinType = LJoinLayout;

if(LLayer_SetParametersEx830(pLayer, &LayerParameters)
== LStatusOK)
{
    // More Processing
    // ...
}

Version

Available in L-Edit 8.2 and later versions. This struct is also available as LLayerParam_Ex00

See Also

LLayerParamEx1512

typedef struct {
    char CIFName[7];
    short GDSNumber; // GDSII layer number. -32,768 <= GDSNumber <= 32,767. The GDSII standard requires 0 <= GDSNumber <= 255.
    short GDSDataType; // GDSII data type. -32,768 <= GDSDataType <= 32,767. The GDSII standard requires 0 <= GDSNumber <= 255. A value of -1 indicates an unassigned GDSII data type.
    double AreaCapacitance; // Area capacitance in aF/Sq. micron from the layer to the substrate. 0 <= AreaCapacitance or AreaCapacitance = -1. A value of -1 indicates an unassigned area capacitance.
    double FringeCapacitance; // Fringe capacitance in fF/micron from the layer to the substrate. 0 <= FringeCapacitance or FringeCapacitance = -1. A value of -1 indicates an unassigned fringe capacitance.
    double FringeCapacitance;
    double Resistivity; // Sheet resistance of the layer in ohms/square. 0 <= Resistivity.
    LBoolean Locked; // Indicates whether the layer is locked.
    LBoolean Hidden; // Indicates whether the layer is hidden.
    LBoolean Protected; // Indicates whether the layer is protected.
    LWireParam WireParam; // Indicates whether the layer is protected.
} LLayerParamEx1512;

Description

LLayerParamEx1512 is an extended layer parameter structure that, unlike LLayerParamEx830, can also process the layer status “protected.”
LLayer_GetParametersEx1512
LLayer_SetParametersEx1512

nd hid
LLayerViewStatus

typedef enum {
    LHidden,
    LVisible
} LLayerViewStatus;

Description

Used to make a layer visible or hidden.

See Also

“LLayerParam” (page 1756)
LLen

typedef unsigned long LLen;

Description

The internal unit used for specifying the magnification ratio.
LMagnification

```c
typedef struct LMagnification {
    LLen num;
    LLen denom;
} LMagnification;
```

**Description**

Specifies the scaling of an object.

**See Also**

“LTransform_Ex99” (page 1802), “Transformation Functions” (page 1673)
**LMarker**

typedef unsigned int LMarker;

**See Also**

“LCell_AddMarker” (page 1315), “LCell_RemoveMarker” (page 1316),
“LCell_RemoveAllMarkers” (page 1317)
typedef struct _LMarkerParam
{
    int StructureSize;      // Use sizeof(LMarkerParam) to assign value to this member
    int EdgeThickness;      // In pixels. Default is 3.
    int EdgeStyle;          // PS_... values for Windows API for Pen GUI objects. Default is 0 for solid.
    int CircleThickness;    // In pixels. Default is 3. 0 means don't show the circle.
    int CrosshairThickness; // In pixels. Default is 3. 0 means don't show the crosshair.
} LMarkerParam;

Description

The parameters that control the appearance of an L-Edit marker.

See Also

“LCell_AddMarker” (page 1315), “LMarkerParam_V11” (page 1767)
LMarkerParam_V11

typedef struct _LMarkerParam_V11
{
    int StructureSize;  // Use sizeof(LMarkerParam) to assign value to
                        // this member
    int EdgeThickness;  // In pixels. Default is 3.
    int EdgeStyle;     // PS_... values for Windows API for Pen GUI
                        // objects.
    Default is 0 for solid.
    int CircleThickness; // In pixels. Default is 3. 0 means don't show
                        // the circle.
    int CrosshairThickness; // In pixels. Default is 3. 0 means don't
                           // show the crosshair.
    unsigned int EdgeColor;  // This is the edge color. Windows COLORREF,
                          // returned by RGB() macro.
    LBooleanGlobal;       // Toggled/removed by "universal" buttons. Used
                          // for DRCEN, SDL, and Node Highlighting.
    unsigned int FillColor;  // This is the fill color. Windows COLORREF,
                          // returned by RGB() macro.
    unsigned short FillPattern[8]; // Fill pattern (make it all 0's for
                                   // no-fill or all 0xff's for solid fill).
    LMarkerStyle Style;
} LMarkerParam_V11;

Description

Parameters that control the appearance of an L-Edit marker.

Use the following settings to control the EdgeStyle. To use EdgeStyle other than 0, the EdgeThickness
must be to set to 1.

- 0 – Solid
- 1 – Dash
- 2 – Dot
- 3 – Dash-Dot
- 4 – Dash-Dot-Dot

Some commonly used color values for setting the EdgeColor:

- Red – 0xFF0000
- Green – 0x00FF00
- Blue – 0x0000FF
- Yellow – 0xFFFF00
- Cyan – 0x00FFFF
- Magenta – 0x0FF00FF
- Black – 0x000000
Example

Example for LCell_AddMarker:

LPoint parr[2];
LMarkerParam_V11 Marker;

parr[0].x = 0;
parr[0].y = 0;
parr[1].x = 1000;
parr[1].y = 0;

memset( &Marker, 0, sizeof(LMarkerParam_V11) );
Marker.StructureSize = sizeof (LMarkerParam_V11);
Marker.Global = LTRUE;
Marker.CircleThickness = 3;
Marker.EdgeColor = 0xFFFF00;
Marker.EdgeThickness = 1;
Marker.EdgeStyle = 1;

LCell_AddMarker (pCell, " ", 2, LFALSE, parr, (LMarkerParam*)&Marker);

See Also

“LMarkerParam” (page 1766), “LCell_AddMarker” (page 1315), “LCell_RemoveMarker” (page 1316)
**LObject**

```c
typedef union _LObject *LObject;
```

**Description**

A pointer to an L-Edit object whose contents can only be accessed or modified through UPI functions.

**See Also**

“Object Functions” (page 1371)
LOrientation

typedef long int LOrientation;
#define LNormalOrientation 0
#define LRotate0 0
#define LRotate90 90
#define LRotate180 180
#define LRotate270 270
#define LRotate0MirrorX -360
#define LRotate90MirrorX -90
#define LRotate180MirrorX -180
#define LRotate270MirrorX -270

Version

The LOrientation data type is made obsolete by the LOrientation_Ex99 data type.

Description

A rotation and/or mirror operation that may be applied to any L-Edit objects.
typedef float LOrientation_Ex99;
#define LNormalOrientation 0
#define LRotate0 0
#define LRotate90 90
#define LRotate180 180
#define LRotate270 270
#define LRotate0MirrorX -360
#define LRotate90MirrorX -90
#define LRotate180MirrorX -180
#define LRotate270MirrorX -270

Description

A rotation and/or mirror operation that may be applied to any L-Edit objects. Rotation can be specified as any real number.

See Also

“LTransform_Ex99” (page 1802).
typedef enum _LOutlineStyle
{
    osUnknown = -1,
    osFirstOutlineStyle = 0,
    osSolid = 0,
    osDotted = 1,
    osShort = 2,
    osShortDot = 3,
    osLongDot = 4,
    osLong = 5,
    osLongDotDot = 6,
    osLongShortShort = 7,
    osLongLongShort = 8,
    osLastOutlineStyle = 8
} LOutlineStyle;

#define NumberOfOutlineStyles (osLastOutlineStyle+1)

Description

This structure defines outline styles. You must use one of the nine predefined outline styles, above; LOutlineStyle is not available to define new styles. The constant NumberOfOutlineStyles represents the total number of predefined outline styles.

See Also

“LRenderingAttribute” (page 1784), “LStipple” (page 1793),
“LLayer_GetRenderingAttribute” (page 1581), “LLayer_SetRenderingAttribute” (page 1582),
“LLayer_GetRenderingObjectName” (page 1583)
LOutlineUnitType

typedef enum _LOutlineUnitType
{
    utPixels = 0,
    utLocatorUnits = 1
} LOutlineUnitType;

typedef unsigned char LStipple[8];

Description

This structure defines the outline unit type as pixels or locator units.
LPalette

typedef LColor LPalette[16];

Description

Gets and sets the color palette of a layout file.

Note:

Note that this data type is superseded in L-Edit V10 and later.

See Also
LPass

typedef struct _LPass *LPass;

Description

A pointer to a layer’s pass list, whose contents can only be accessed or modified through UPI functions.

See Also

“LPass_New” (page 1576), “LPass_GetList” (page 1577)
**LPassMode**

```c
typedef unsigned char LStipple[8];
typedef enum {
    LSet=16,
    LClear=8
} LPassMode;
```

**Description**

*LPassMode* is used to specify the write mode of a pass (set or clear).

**Note:**

Note that this data type is superseded in L-Edit V10 and later.

**See Also**

“*LPassParam*” (page 1777)
### LPassParam

```c
typedef struct _LPassParam {
    unsigned char ColorIndex;
    LPassMode WriteMode;
    LStipple Stipple;
} LPassParam;
```

**Description**

Specifies the properties of a pass, including its color index, pass mode, and stipple pattern.

**Note:**

Note that this function is superseded in L-Edit V10 and later.

**See Also**

“LPassMode” (page 1776).
LPassType

typedef enum {
    LObjectPass,
    LPortPass,
    LTextPass
} LPassType;

Description

Specifies the type of a pass (object, port, or text).

Note:  
Note that this function is superseded in L-Edit V10 and later.

See Also

“LPass_GetList” (page 1577)
LPieParams

typedef struct
{
    LPoint ptCenter;
    LCoord nRadius;
    double dStartAngle;
    double dStopAngle;
} LPieParams

Description

Specifies the properties of a pie object, including its center, radius, start angle, and stop angle. The valid ranges for pie parameters are:

- \(-\text{WORLD\_MAX} \leq \text{Center.x} \leq \text{WORLD\_MAX}\)
- \(-\text{WORLD\_MAX} \leq \text{Center.y} \leq \text{WORLD\_MAX}\)
- \(0 < \text{Radius} \leq \text{WORLD\_MAX}\)
- \(0 \leq \text{StartAngle} \leq 360\)
- \(0 \leq \text{StopAngle} \leq 360\)

See Also

“Torus and Pie Functions” (page 1421)
LPoint

typedef struct {
    LCoord y, x;
} LPoint;

Description

A point in the L-Edit two-dimensional layout space.

See Also

“Point Functions” (page 1663)
LPort

typedef struct _LPort *LPort;

Description

A pointer to an L-Edit port whose contents can only be accessed or modified through UPI functions. A port is a text whose location in a cell is specified by a rectangle, and it is typically used for documentation or routing purposes. Each cell has a single list of ports. Each port has a layer, text, and a location associated with it.

See Also

“Port Functions” (page 1452)
LPropertyType

typedef enum
{
    L_unassigned=0,
    L_none,
    L_int, // int
    L_real, // double
    L_bool, // 0 if false, otherwise true
    L_string, // char *
    L_enum, // not used
    L_byte, // byte
    L_ptr,     // void *
    L_blob     // void *
} LPropertyType;

Description

An enumerated datatype indicating the type of an L-Edit property.

See Also

“Entity Functions” (page 1345)
LRect

typedef struct {
   LCoord y0, x0;
   LCoord y1, x1;
} LRect;

Description

The coordinates of a rectangle in layout space. Here, (x0, y0) is the lower left corner of a rectangle and (x1, y1) is the upper right corner.

See Also

“Rectangle Functions” (page 1669)
LRenderingAttribute

typedef enum _LRenderingMode
{
    rmPaint,
    rmAdd,
    rmSubtract
} LRenderingMode;

typedef enum _LOutlineUnitType
{
    utPixels = 0,
    utLocatorUnits = 1
} LOutlineUnitType;

typedef enum _LRenderingAttributeIndex
{
    raiFirstRenderingAttribute = 0,
    raiObject = 0,
    raiSelectedObject = 1,
    raiPortBox = 2,
    raiSelectedPortBox = 3,
    raiPortText = 4,
    raiSelectedPortText = 5,
    raiWireCenterline = 6,
    raiSelectedWireCenterline = 7,
    raiLastRenderingAttribute = 7
} LRenderingAttributeIndex;

#define NumberOfRenderingAttributes (raiLastRenderingAttribute+1)

typedef enum
{
    LMIN_WIDTH,
    LEXACT_WIDTH,
    LOVERLAP,
    LEXTENSION,
    LNOT_EXISTS,
    LSPACING,
    LSURROUND,
    LDENSITY
} LDrCRuleType;

typedef struct _LRenderingAttribute
{
    LRenderingMode mMode; /* rmPaint=draw, rmAdd=OR with background, 
    rmSubtract=AND with background */
    unsigned int mPass; /* Pass number from 1 to 10 */
    LStipple mFillPattern; /* Pattern */
    unsigned int mFillColorIndex; /* Color */
    LStipple mOutlinePattern; /* Outline pattern */
    unsigned int mOutlineColorIndex; /* Outline color */
    LOutlineStyle mOutlineStyle; /* Outline style */
    LOutlineUnitType mOutlineWidthUnits; /* utPixels or utLocatorUnits. 
    utPixels is recommended for better performance */

unsigned int mOutlineWidth; /* 1 is recommended for best performance */
unsigned char mbBGprevPass;
}
LRenderingAttribute;

typedef LRenderingAttribute *LLRenderingAttribute;

Description

Defines rendering attributes.

See Also

“LStipple” (page 1793), “LLayer_GetRenderingObjectName” (page 1583),
“LLayer_GetRenderingAttribute” (page 1581), “LLayer_SetRenderingAttribute” (page 1582),
“LFile_GetColorPalette” (page 1602)
**LRenderingAttributeIndex**

```c
typedef enum _LRenderingAttributeIndex
{
    raiFirstRenderingAttribute = 0,
    raiObject = 0,
    raiPortBox = 1,
    raiPortText = 2,
    raiWireCenterline = 3,
    raiSelectedObject = 4,
    raiSelectedPortBox = 5,
    raiSelectedPortText = 6,
    raiSelectedWireCenterline = 7,
    raiLastRenderingAttribute = 7
} LRenderingAttributeIndex;
```

#define NumberOfRenderingAttributes (raiLastRenderingAttribute+1)

**Description**

Lists the available rendering attributes. The constant `NumberOfRenderingAttributes` represents the total number of defined rendering attributes.

**See Also**

LRendingMode

typedef enum _LRencodingMode
{
    rmPaint = 0,
    rmAdd = 1,
    rmSubtract = 2
}
LRencodingMode;

Description

Lists the type of rendering process.
typedef struct _LSelection *LSelection;

Description

A pointer to the L-Edit selection list whose contents can only be accessed or modified through UPI functions.

See Also

“Selection Functions” (page 1494)
LSelectionParam

typedef struct _LSelectionParam {
  long selection_range;
  long deselect_distance_2;
  long deselect_distance_1;
  long lambda_edit_range;
  long pixel_edit_range;
  int select_draws;
} LSelectionParam;

typedef enum
{
  LSnapping,
  LSmooth
} LCursorType;

Description

Used to get and set the selection setup of file. This structure is used to specify the selection range,
deselection range, and the edit range. The switch select_draws determines if an object will be
automatically selected after it is created.

See Also

“LFile_GetSelectionParam” (page 1234), “LFile_SetSelectionParam” (page 1235)
LShapeType

typedef enum {
    LBox,
    LCircle,
    LWire,
    LPolygon,
    LTorus,
    LPie,
    LOtherObject,
    LObjInstance,
    LObjPort,
    LObjRuler
} LShapeType;

Description

An enumeration of the object type of an L-Edit object.

See Also

“LObject_GetShape” (page 1380)
LSpecialLayer

typedef enum {
    GridLayer,
    OriginLayer,
    CellOutlineLayer,
    ErrorLayer,
    IconLayer,
    FirstMaskLayer,
    DragBoxLayer
} LSpecialLayer;

Description

An enumerated datatype that specifies the type of a special layer.

See Also

“Layer Functions” (page 1528)
typedef enum {
    LStatusOK = 0,
    LTooManyInits = 1,
    LOpenError = 2,
    LCloseError = 3,
    LCreateError = 4,
    LSsaveError = 5,
    LBadFile = 6,
    LBadCell = 7,
    LBadLayer = 8,
    LBadParameters = 9,
    LBadObject = 10,
    LBadHierarchy = 11,
    LTmError = 12,
    LUserDataError = 13,
    LCellOverWritten = 14,
    LLayerMapsDifferent = 15,
    LNamedCellExists = 16,
    LCopyProtViolation = 17,
    LNoSelection = 18,
    LVertexNotFound = 19,
    LCantDeleteVertex = 20,
    LPropertyNotFound = 21,
    LPropertyHasNoValue = 22,
    LPropertyTypeMismatch = 23,
    LBufferTooSmall = 24,
    LSystemError = 25,  // (system errors like cannot load DLL)
    LPropertyConversionError = 26, // (an error occurred during the export or import of properties to/from a file)
    LUserAbort = 27, // (user aborted the operation such as DRC, Extract, etc.)
    LExists = 28, // (item added already exists - LDrcRule, LLayer, etc.)
    LLayerNotEmpty = 29, // (layer has objects on it)
    LParameterOutOfRange = 30,
    LMaxRunTimeExceeded = 31  // (for DRC Automation)
} LStatus;

Description

LStatus is an enumeration of various error returns. A return value of zero indicates no errors; a value greater than zero indicates an error by its position in the list.
**LStipple**

typedef unsigned char LStipple[8];

**Description**

Each unsigned character in the array **LStipple** is a bitmap of a row in the 8x8 stipple pattern.

The **LStipple** is a one dimensional 8 bit character array. Each character in the array is a bitmap of a row in the 8x8 stipple pattern. Each character value is represented by an 8 bit binary value. L-Edit uses a black box to represent the binary value 1 and a white box to represent the binary value 0.

In the following example, the following Fill Pattern is assigned to the Metal1 layer.

```c
ra.mFillPattern[0] = 1; // 00000001 = 0x01
ra.mFillPattern[1] = 2; // 00000010 = 0x02
ra.mFillPattern[2] = 4; // 00000100 = 0x04
ra.mFillPattern[3] = 8; // 00001000 = 0x08
ra.mFillPattern[4] = 16; // 00010000 = 0x10
ra.mFillPattern[5] = 32; // 00100000 = 0x20
ra.mFillPattern[6] = 64; // 01000000 = 0x40
ra.mFillPattern[7] = 128; // 10000000 = 0x80
```

The resulting bitmap is shown below.

![Create Pattern](image)

If you assign an ASCII character instead of a number to each bit of the character array, the ASCII character will be converted to a binary value according to the ASCII conversion table. In L-Edit, you can display your design in 16, 32, 64, 128 or 256 colors in your palette. For 256 colors, you can use an 8 bit binary value to represent a color as you would use a 4 bit binary value to represent 16 colors.
For example, in a 256 color palette, the green color index is 00010001, which is equivalent to decimal number 17 or hexadecimal number 0x11. In the screenshot below, the black dot represents binary value 1 and the white dot represents binary value 0 (read from right to left).

![Color Index Example](image)

In a 16 color palette the green color index is 0010, which is equivalent to decimal number 2 or hexadecimal number 0x2, as shown in the screenshot below.

![Color Index Example](image)

**Example**

**See Also**

“LRenderingAttribute” (page 1784).
tech_unit_type

Description

Specifies the units of technology measurement.

```c
typedef enum {
    MICRONS,
    MILLIMETERS,
    CENTIMETERS,
    MILS,
    INCHES,
    LAMBDA,
    OTHER
} tech_unit_type;
```

See Also

“Technology Setup Functions” (page 1590)
LTCellParameterType

Description

An enumerated datatype used to get and set the parameters of a T-Cell. If a T-Cell is specified, gets the default value of the parameter. If a cell generated from a T-Cell is specified, gets the parameter value used to generate the cell.

```c
typedef enum
{
   L_notexist=-1,
   L_bool,
   L_int,
   L_float,
   L_string,
   L_layer,
   L_hstretch,
   L_vstretch
} LTCellParameterType;
```

Version

This datatype replaces the function “LCell_GetParameter” (page 1309).

See Also

“LCell_GetParameterAsString” (page 1701), “LCell_GetParameterName” (page 1702), “LCell_GetParameterType” (page 1703),
LTechnology

Description

A structure where information about the technology of the design file can be stored.

typedef struct _LTechnology {
    const char* name; /*Technology name*/
    tech_unit_type unit_type; /*Unit of measurement*/
    const char* unit_name; /*Other unit name*/
    long num; /*Numerator of mapping*/
    long denom; /*Denominator of mapping*/
    long lambda_num; /*Numerator, lambda mapping*/
    long lambda_denom; /*Denominator, Lambda mapping*/
} LTechnology;

Version

This data type is deprecated by LTechnologyEx840
LTechnologyEx840

Description

typedef struct _LTechnologyEx840
{
    char szName[128];
    tech_unit_type eUnitType;
    char szUnitName[128];
    long nNum;
    long nDenom;
    long nOtherNum;
    long nOtherDenom;
} LTechnologyEx840;

See Also

“Technology Setup Functions” (page 1590)
LTerminalType

typedef enum

typedef struct _LTechnologyEx840
{
    TT_In = 0,
    TT_Out,
    TT_Inout,
    TT_Switch,
    TT_Jumper,
    TT_Unused,
    TT_Tristate
} LTerminalType;

Description

LTerminalType is an enumerated datatype used to specify the Terminal type of a port.

Port Access Direction

You should also define the port access direction, which uses type int. The following bits are used.

#define PORT_ACCESS_DIRECTION_TOP        1
#define PORT_ACCESS_DIRECTION_LEFT       2
#define PORT_ACCESS_DIRECTION_RIGHT      4
#define PORT_ACCESS_DIRECTION_BOTTOM     8

See Also

“LLabel” (page 1754), “Port Functions” (page 1452).
LTorusParams

typedef struct
{
    LPoint ptCenter;
    LCoord nInnerRadius;
    LCoord nOuterRadius;
    double dStartAngle;
    double dStopAngle;
} LTorusParams;

Description

Specifies the parameters of a torus, including the center point, inner and outer radii, and start and stop angles. The valid ranges for torus parameters are given below, where [ ] indicate included endpoints and ( ) indicate excluded endpoints.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptCenter.x</td>
<td>[-WORLD_MAX, WORLD_MAX]</td>
</tr>
<tr>
<td>ptCenter.y</td>
<td>[-WORLD_MAX, WORLD_MAX]</td>
</tr>
<tr>
<td>nInnerRadius</td>
<td>(0, WORLD_MAX]</td>
</tr>
<tr>
<td>nOuterRadius</td>
<td>(0, WORLD_MAX]</td>
</tr>
<tr>
<td>dStartAngle</td>
<td>[0, 360]</td>
</tr>
<tr>
<td>dStopAngle</td>
<td>[0, 360]</td>
</tr>
</tbody>
</table>

See Also

"Torus and Pie Functions" (page 1421)
LTransform

typedef struct {
    LPoint translation;
    LOrientation orientation;
    LMagnification magnification;
} LTransform;

Description

Specifies the translation, orientation, and magnification of an object. All objects, ports, and instances can be transformed.

Note:

Note that this function is obsolete in L-Edit V10 and later.

See Also

LTransform_Ex99

typedef struct {
    LPoint translation;
    LOrientation_Ex99 orientation;
    LMagnification magnification;
} LTransform_Ex99;

Description

Specifies the translation, orientation as a real number, and magnification of an object. All objects, ports, and instances can be transformed.

See Also

LVertex

typedef struct _LVertex *LVertex

Description

A pointer to a polygon’s vertex.

Version

Available in L-Edit 8.4 and later versions.

See Also

“Vertex Functions” (page 1396).
LWindow

typedef struct _LWindow *LWindow;

Description

A pointer to an L-Edit window whose contents can only be accessed or modified through UPI functions.

See Also

“Interface Functions” (page 1075), “Windows Functions” (page 1128)
LWindowType

```c
typedef enum {
    UNKNOWN = 0,
    CELL_BROWSER,
    TEXT,
    LAYOUT,
    CROSS_SECTION,
    CODE,
    LW_SPICE,
    LW_COMMAND,
    LW_HTML,
    LW_LOG
} LWindowType;
```

Description

Enumeration of different window types that can be opened in L-Edit.

Enumerators

- **UNKNOWN**: Unknown window type
- **CELL_BROWSER**: Design Navigator window
- **TEXT**: General text window
- **LAYOUT**: Cell layout window
- **CROSS_SECTION**: Layout window that is displaying a cross-section
- **CODE**: A text window that is either a T-Cell Code window or a UPI macro code window (.cpp file). Code windows have syntax highlighting for C++ and UPI functions.
- **LW_SPICE**: A text window that contains a SPICE file (.sp, .spc, or .cir files). SPICE windows have syntax highlighting for SPICE commands.
- **LW_LOG**: A text window that contains a log file (.log file). Log windows have syntax highlighting for log files (Error are in red, Warnings are in blue).
- **LW_COMMAND**: A text window that contains a Calibre, Assura or Dracula DRC command file.
- **LW_HTML**: A text window in HTML format, such as the DRC Summary Report.

See Also

“Interface Functions” (page 1075), “Windows Functions” (page 1128), “LWindow_GetType” (page 1139)
LWireConfig

typedef struct {
    LCoord width;
    LJoinType join;
    LCapType cap;
    LCoord miter_angle;
} LWireConfig;

Description

Specifies the configuration of a wire. The configuration of a wire includes width, join type, cap type, and miter angle.

See Also

“LWire_New” (page 1433)
LWireConfigBits

typedef enum {
    LSetWireWidth = 1 << 0,
    LSetWireJoin = 1 << 1,
    LSetWireCap = 1 << 2,
    LSetWireMiterLimit = 1 << 3,
    LSetWireAll = -1
} LWireConfigBits;

Description

Used to mask out configuration properties that you do not wish to set.

See Also

“LWireConfig” (page 1806), “L Wire New” (page 1433)
LWireParam

typedef struct {
    long  defaultWireWidth;
    short defaultWireMiterAngle;
    LCapType  capType;
    LJoinType  joinType;
} LWireParam;

Description

Specifies the default properties of a wire: wire width, miter angle, join style, and end style.

See Also

“LLayer_SetParametersEx830” (page 1545), “LWire_New” (page 1433)
typedef enum
{
    LSelectionTool = 0; /*Selection tool*/
    LBoxTool; /*Box tool*/
    LPolygon90Tool; /*Orthogonal polygon tool*/
    LPolygon45Tool; /*45 degree polygon tool*/
    LPolygonAAtool; /*All angle polygon tool*/
    LWire90Tool; /*Orthogonal wire tool*/
    LWire45Tool; /*45 degree wire tool*/
    LWireAAtool; /*All angle wire tool*/
    LCircleTool; /*Circle tool*/
    LPieWedgeTool; /*Pie wedge tool*/
    LTorusTool; /*Torus tool*/
    LPortTool; /*Port tool*/
    LRuler90Tool; /*Orthogonal ruler tool*/
    LRuler45Tool; /*45 degree tool*/
    LRulerAAtool; /*All angle tool*/
    LInstanceTool; /*Instance tool (Not currently implemented*/
} UPIDrawingToolType;
Introduction

LComp is a set of high-level C functions for L-Edit’s User Programmable Interface (UPI). LComp functions provide a means to easily create and position instances of cells, add cell geometry, and perform other basic cell operations with simple programming.

Functions are arranged in five primary categories:

- **“State Functions”** (page 1814)
- **“Placement Functions”** (page 1835)
- **“Position Functions”** (page 1842)
- **“Geometry Functions”** (page 1849)
- **“Utility Functions”** (page 1859)

Composition with LComp

One of the key features of LComp is a set of functions that facilitate composition, the placement and interconnection of cells in the layout. Because such composition often requires large-scale repetition of a layout scheme, LComp functions use a set of state variables to determine how and where consecutive instances will be placed. Once state variables are initialized, you can begin instancing and placing cells without repeating key parameter values in each function call.

LComp **“Placement Functions”** (page 1835) allow you to easily create and/or place instances in the active cell. Cells, instances, and ports are passed to LComp functions as elements (see “Elements,” below). There are two basic ways to place an element in the layout:

- You can position the element such that its reference point is placed at the current \((x, y)\) position.
- You can position the element so that it lines up with an edge of the last placed element.

Each method uses the values of global state variables to determine factors such as instance boundaries, orientation, and, composition direction. See “State Variables” on page 1813 for an explanation of these parameters.

Initializing LComp

Before using LComp functions, you must first call an initialization function as follows:

```
LC_InitializeState();
```

**LC_InitializeState** is required to get handles to the currently open file and the currently open cell. Composition functions that follow (such as placing an instance or creating geometry) are executed in the current cell. **LC_InitializeState** also obtains the default abutment layer (see “State Variables” on page 1813).
Elements

LComp operations are performed on elements. An LComp element can be a cell, an instance, a port, or a hierarchical combination of cell, instance, and/or port.

The general format for naming LComp elements is:

cellname/inistance/port

where a slash (/) is used to separate levels of hierarchy. In addition, the following rules apply:

- If an instance name is *not* unique within the design, then you must refer to the instance name as *source_cell:instance*
- A port may optionally be specified with the corresponding layername: *port:layername*
- If the first component of a name is a cell name, the result is in the context of that cell; otherwise, it is in the context of the current cell.

For example, the element *Top/Bottom:BotInst/Out* refers to the port named *Out* in instance of cell *Bottom* named *BotInst*, located in cell *Top*. The element *Top/Bottom:BotInst/Out* is illustrated below:

The instance named *LastInstance* refers to the last instance placed using one of the “Placement Functions” (page 1835).

*Note:* To refer to an instance or port in an element name, you must use the associated instance or port name. Unnamed instances and ports are inaccessible as elements.
State Variables

LComp functions use a core set of state variables to determine how and where instances are placed in the layout. The state variables and the key functions used to set them are listed in the following table. See “State Functions” (page 1814) for additional functions that set or retrieve state variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
<th>Key Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>placement position</td>
<td>The ((x, y)) coordinates at which the next instance or port will be placed. LComp has an implicit composition capability; after an instance is placed in the layout, the placement position increments automatically by the width or height of the instance’s bounding box. The direction in which the position is incremented is determined by the composition direction (see “LC_SetCompositionDirection” (page 1823)).</td>
<td>“LC_SetXYPlacementPosition” (page 1827)</td>
</tr>
<tr>
<td>composition direction</td>
<td>The location in which an instance is placed relative to the last placed instance. You can build a layout composition in any of four directions - up, down, left, or right.</td>
<td>“LC_SetCompositionDirection” (page 1823)</td>
</tr>
<tr>
<td>reference point</td>
<td>The location within an instance that corresponds to its ((x, y)) position in the layout.</td>
<td>“LC_SetReferencePoint” (page 1815)</td>
</tr>
<tr>
<td>abutment type</td>
<td>Determines how cell boundaries are defined. Instances can be aligned by either their minimum bounding boxes or by their abutment boxes. The abutment box is the smallest box enclosing all objects on the abutment layer. The default abutment layer is the Icon special layer in L-Edit.</td>
<td>“LC_SetAbutmentType” (page 1817)</td>
</tr>
<tr>
<td>orientation</td>
<td>The transformation (rotation and/or reflection) applied to instances when they are placed in the layout.</td>
<td>“LC_SetPlacementOrientation” (page 1819)</td>
</tr>
<tr>
<td>overlap</td>
<td>The amount by which adjacent instance boundaries overlap. Boundary definitions are controlled by the abutment type (see “LC_SetAbutmentType” (page 1817)).</td>
<td>“LC_SetPlacementOverlap” (page 1825)</td>
</tr>
</tbody>
</table>
State Functions

LComp functions use a core set of global state variables to determine how and where instances are placed in the layout. State functions allow you to set and retrieve state variable values.

“LC_SetReferencePoint” (page 1815)
“LC_SetAbutmentType” (page 1817)
“LC_SetPlacementOrientation” (page 1819)
“LC_AddPlacementOrientation” (page 1821)
“LC_SetCompositionDirection” (page 1823)
“LC_SetPlacementOverlap” (page 1825)
“LC_SetXYPlacementPosition” (page 1827)
“LC_SetXPlacementPosition” (page 1829)
“LC_SetYPlacementPosition” (page 1831)
“LC_IncrementXPlacementPosition” (page 1833)
“LC_IncrementYPlacementPosition” (page 1833)
“LC_GetReferencePoint” (page 1816)
“LC_GetAbutmentType” (page 1818)
“LC_GetPlacementOrientation” (page 1820)
“LC_SubtractPlacementOrientation” (page 1822)
“LC_GetCompositionDirection” (page 1824)
“LC_GetPlacementOverlap” (page 1826)
“LC_GetXYPlacementPosition” (page 1828)
“LC_GetXPlacementPosition” (page 1830)
“LC_GetYPlacementPosition” (page 1832)
LC_SetReferencePoint

RelativeTo LC_SetReferencePoint(RelativeTo place);

Description

Sets the current reference point for LComp placement and position functions. The reference point can be the origin of the source cell’s coordinate system, or one of nine points on the object’s bounding box:

Return Value

Returns the previous reference point.

Parameters

The reference point place can be set to any of the nine points illustrated above (UL, TOP, UR, etc.). When the value ORIGIN is used, the reference point is the origin of the source (instanced) cell’s coordinate system.

See Also

“LC_GetReferencePoint” (page 1816), “LC_SetAbutmentType” (page 1817), “RelativeTo” (page 1879)
LC_GetReferencePoint

RelativeTo LC_GetReferencePoint(void);

Description

Returns the current reference point for LComp placement and position functions.

See Also

“LC_SetReferencePoint” (page 1815), “LC_SetAbutmentType” (page 1817), “RelativeTo” (page 1879)
LC_SetAbutmentType

AbutTo LC_SetAbutmentType(AbutTo place);

Description

Determines how a cell’s bounding box is defined in LComp placement and position functions.

Return Value

Returns the previous abutment type.

Parameters

The parameter place can be set to either MBB (minimum bounding box) or ABUT.

When MBB is specified, the bounding box of a cell is the smallest box that encloses all geometry in the cell, ignoring the text portions of labels. When ABUT is specified, the bounding box of a cell is the smallest box that encloses all geometry on the abutment layer, ignoring the text portions of labels. The default abutment layer is the Icon special layer in L-Edit; you can change the abutment layer using LC_SetAbutmentLayer.

Note:

A port is always bounded by the port box.

See Also

“LC_GetAbutmentType” (page 1818), “AbutTo” (page 1877)
**LC_GetAbutmentType**

`AbutTo LC_GetAbutmentType(void);`

**Description**

Returns the current abutment type.

**See Also**

“LC_SetAbutmentType” (page 1817), “AbutTo” (page 1877)
LC_SetPlacementOrientation

LOrientation LC_SetPlacementOrientation(LOrientation orient);

Description

Sets the orientation (rotation) in which instances or ports are placed.

Return Value

Returns the previous orientation setting.

Parameters

The parameter orient specifies a transformation (rotation and/or reflection) of the instance. Only the following values are supported:

- LNormalOrientation: No transformation.
- LRotate90: 90 degree rotation.
- LRotate180: 180 degree rotation.
- LRotate270: 270 degree rotation.
- LRotate0MirrorX: Flips the instance horizontally (reflection).
- LRotate90MirrorX: Flips horizontally, then applies a 90 degree rotation.
- LRotate180MirrorX: Flips horizontally, then applies a 180 degree rotation.
- LRotate270MirrorX: Flips horizontally, then applies a 270 degree rotation.

See Also

**LC_GetPlacementOrientation**

```c
LOrientation LC_GetPlacementOrientation(void);
```

**Description**

Returns the current orientation for placing instances.

**See Also**

LC_AddPlacementOrientation

LOrientation LC_AddPlacementOrientation(LOrientation orig, LOrientation new_orientation);

Description

Computes the equivalent transformation obtained by following the transformation specified by orig with the transformation specified by new_orientation.

Return Value

Returns the new orientation.

Parameters

The parameters orig and new_orientation are the first and second transformations, respectively. Only the following transformations are supported:

- LNormalOrientation: No transformation.
- LRotate90: 90 degree rotation.
- LRotate180: 180 degree rotation.
- LRotate270: 270 degree rotation.
- LRotate0MirrorX: Flips the instance horizontally (reflection).
- LRotate90MirrorX: Flips horizontally, then applies a 90 degree rotation.
- LRotate180MirrorX: Flips horizontally, then applies a 180 degree rotation.
- LRotate270MirrorX: Flips horizontally, then applies a 270 degree rotation.

See Also

LC_SubtractPlacementOrientation

LOrientation LC_SubtractPlacementOrientation(LOrientation orig, LOrientation new_orientation);

Description

Computes the equivalent transformation obtained by first executing the transformation specified by
orig, then reversing the transformation specified by new_orientation.

Return Value

Returns the new orientation.

Parameters

The parameters orig and new_orientation are the original and subtracted transformations, respectively.
Only the following transformations are supported:

- LNormalOrientation: No transformation.
- LRotate90: 90 degree rotation.
- LRotate180: 180 degree rotation.
- LRotate270: 270 degree rotation.
- LRotate0MirrorX: Flips the instance horizontally (reflection).
- LRotate90MirrorX: Flips horizontally, then applies a 90 degree rotation.
- LRotate180MirrorX: Flips horizontally, then applies a 180 degree rotation.
- LRotate270MirrorX: Flips horizontally, then applies a 270 degree rotation.

See Also

“LC_SetPlacementOrientation” (page 1819), “LC_GetPlacementOrientation” (page 1820),
“LC_AddPlacementOrientation” (page 1821), “LOrientation” (page 1770)
LC_SetCompositionDirection

CompositionDirectionType LC_SetCompositionDirection(CompositionDirectionType direction);

Description

Specifies the direction in which LComp places instances next to each other.

Return Value

Returns the previous composition direction.

Parameters

The parameter direction takes one of the following values:

- UP (or VERTICAL)
- DOWN
- LEFT
- RIGHT (or HORIZONTAL)
- NONE

See Also

“LC_GetCompositionDirection” (page 1824), “CompositionDirectionType” (page 1878)
LC_GetCompositionDirection

```
CompositionDirectionType LC_GetCompositionDirection(void);
```

Description

Returns the current composition direction.

See Also

“LC_SetCompositionDirection” (page 1823), “CompositionDirectionType” (page 1878)
LC_SetPlacementOverlap

```c
LCoord LC_SetPlacementOverlap(LCoord overlap);
```

**Description**

Sets the placement overlap between instance or port boundaries in LComp positioning functions.

**Return Value**

Returns the previous placement overlap.

**Parameters**

The parameter *overlap* specifies a distance, in internal units, by which to overlap elements in either the x- or y-direction. The direction of overlap (x or y) depends on the composition direction.

**See Also**

“LC_GetPlacementOverlap” (page 1826), “LC_SetCompositionDirection” (page 1823), “LCoord” (page 1707)
**LC_GetPlacementOverlap**

```c
LCoord LC_GetPlacementOverlap(void);
```

**Description**

Returns the current placement overlap, in internal units, between instance or port boundaries in LComp positioning functions.

**See Also**

“LC_GetPlacementOverlap” (page 1826), “LC_SetCompositionDirection” (page 1823), “LCoord” (page 1707)
LC_SetXYPlacementPosition

LPoint LC_SetXYPlacementPosition(LCoord xpos, LCoord ypos);

Description

Sets the current (x, y) placement position to the point (xpos, ypos), where xpos and ypos are in internal units.

See Also

“LC_GetXYPlacementPosition” (page 1828), “LC_SetXPlacementPosition” (page 1829),
“LC_GetXPlacementPosition” (page 1830), “LC_SetYPlacementPosition” (page 1831),
“LC_GetYPlacementPosition” (page 1832), “LC_IncrementXPlacementPosition” (page 1833),
“LC_IncrementYPlacementPosition” (page 1834)
LC_GetXYPlacementPosition

LPoint LC_GetXYPlacementPosition(void);

Description

Returns the current (x, y) placement position in internal units.

See Also

“LC_SetXYPlacementPosition” (page 1827), “LC_SetXPlacementPosition” (page 1829),
“LC_GetXPlacementPosition” (page 1830), “LC_SetYPlacementPosition” (page 1831),
“LC_GetYPlacementPosition” (page 1832), “LC_IncrementXPlacementPosition” (page 1833),
“LC_IncrementYPlacementPosition” (page 1834)
**LC_SetXPlacementPosition**

LCoord LC_SetXPlacementPosition(LCoord newx);

**Description**

Sets the x-coordinate of the (x, y) placement position to newx (internal units).

**See Also**

**LC_GetXPlacementPosition**

```c
LCoord LC_GetXPlacementPosition(void);
```

**Description**

Returns the x-coordinate of the current (x, y) placement position in internal units.

**See Also**

“LC_SetXYPlacementPosition” (page 1827), “LC_GetXYPlacementPosition” (page 1828),
“LC_SetXPlacementPosition” (page 1829), “LC_SetYPlacementPosition” (page 1831),
“LC_GetYPlacementPosition” (page 1832), “LC_IncrementXPlacementPosition” (page 1833),
“LC_IncrementYPlacementPosition” (page 1834)
LC_SetYPlacementPosition

```
LCoord LC_SetYPlacementPosition(LCoord newy);
```

Description

Sets the y-coordinate of the (x, y) placement position to `newy` (internal units).

See Also

LC_GetYPlacementPosition

typedef int LC_GetYPlacementPosition();

Description

Returns the y-coordinate of the current (x, y) placement position.

See Also

“LC_SetXYPlacementPosition” (page 1827), “LC_GetXYPlacementPosition” (page 1828),
“LC_SetXPlacementPosition” (page 1829), “LC_GetXPlacementPosition” (page 1830),
“LC_SetYPlacementPosition” (page 1831), “LC_IncrementXPlacementPosition” (page 1833),
“LC_IncrementYPlacementPosition” (page 1834)
LC_IncrementXPlacementPosition

```c
LCoord LC_IncrementXPlacementPosition(LCoord newx);
```

Description

Increments the x-coordinate of the current (x, y) placement position by `newx` internal units.

See Also

“LC_SetXYPlacementPosition” (page 1827), “LC_GetXYPlacementPosition” (page 1828),
“LC_GetXPlacementPosition” (page 1830), “LC_SetYPlacementPosition” (page 1831),
“LC_GetYPlacementPosition” (page 1832), “LC_IncrementYPlacementPosition” (page 1834)
**LC_IncrementYPlacementPosition**

```c
LCoord LC_IncrementYPlacementPosition(LCoord newy);
```

**Description**

Increments the y-coordinate of the current (x, y) placement position by `newy` internal units.

**See Also**

“LC_SetXYPlacementPosition” (page 1827), “LC_GetXYPlacementPosition” (page 1828),
“LC_GetXPlacementPosition” (page 1830), “LC_SetYPlacementPosition” (page 1831),
“LC_GetYPlacementPosition” (page 1832), “LC_IncrementXPlacementPosition” (page 1833)
Placement Functions

Placement functions allow you to easily create and/or place instances in the active cell. The position and orientation of instances is determined by the state variables (see “State Functions” on page 1814).

“LC_Position” (page 1836)       “LC_Instance” (page 1837)
“LC_Generate” (page 1838)        “LC_Align” (page 1839)
“LC_InstanceAlign” (page 1840)    “LC_GenerateAlign” (page 1841)
LC_Position

LStatus LC_Position(char *element);

Description

Places an instance in the layout by placing the instance, a subcell, or a port at the current \((x, y)\) position. If a cell name is passed as an argument, LC_Position creates an instance of that cell, then places it. After placing the instance, LC_Position updates the \((x, y)\) placement position according to the current composition direction.

Note: To create a named instance of a cell, use “LC_Instance” (page 1837). An instance name is required if you wish to refer to the instance again in LComp functions.

Return Value

LStatusOK if successful. If an error occurs, returns the error value.

Parameters

The parameter element is the name of a cell, instance, or port (e.g., “Topcell/InstA/PortB”). See “Elements” on page 1812 for an explanation of element names.

See Also

“LC_Instance” (page 1837)
LC_Instance

LInstance LC_Instance(char *classname, char *instname);

Description

Creates an instance named instname of the cell classname, then places the instance in the layout at the current (x,y) position. After placing the instance, LC_Instance updates the (x, y) placement position according to the current composition direction.

Note: To create an instance of a parameterized T-Cell, use “LC_Generate” (page 1838).

Return Value

Returns a pointer to the newly created instance if successful. If an error occurs, returns NULL.

Parameters

classname Name of the cell to be instanced.
instname Name of the newly created instance.

See Also

“LC_Position” (page 1836)
LC_Generate

\[
\text{LInstance } \text{LC\_Generate}(\text{char } *\text{classname, char } *\text{instname, char } **\text{params});
\]

Description

Creates an instance named \textit{instname} of the T-Cell named \textit{classname}, using the T-Cell parameter values specified by \textit{params}. After placing the instance, \texttt{LC\_Generate} updates the \((x, y)\) placement position according to the current composition direction. This is similar to “\texttt{LC\_Instance}” (page 1837), with the addition of T-Cell parameter inputs.

Return Value

Returns a pointer to the newly created instance if successful. Otherwise, returns NULL.

Parameters

- \textit{classname} Name of the T-Cell to be instanced.
- \textit{instname} Name of the newly created instance.
- \textit{params} T-Cell parameter values.

See Also

“\texttt{LC\_GenerateAlign}” (page 1841)
LC_Align

LInstance LC_Align(char *element);

Description

Aligns the specified instance, subcell, or port next to the last placed instance. If a cell name is passed as an argument, LC_Position creates an instance of that cell, then places it. After placing the instance, LC_Align updates the (x, y) placement position according to the current composition direction.

Note: To create and align a named instance of a cell, use “LC_InstanceAlign” (page 1840). An instance name is required if you wish to refer to the instance again in LComp functions.

Return Value

Returns a pointer to the instance if successful. If an error occurs, returns NULL.

Parameters

The parameter element is the name of a cell, instance, or port (e.g., “Topcell/InstA/PortB”). See “Elements” on page 1812 for an explanation of element names.

See Also

“LC_InstanceAlign” (page 1840)
LC_InstanceAlign

LInstance LC_InstanceAlign(char *classname, char *instname);

Description

Creates an instance named instname of the cell named classname, and aligns it next to the last placed instance. This function also increments the current (x,y) position according to the current composition direction.

Return Value

Returns a pointer to the newly created instance if successful. If an error occurs, returns NULL.

Parameters

classname  Name of the cell to be instanced.
instname  Name of the newly created instance.

See Also

“LC_Align” (page 1839)
LC_GenerateAlign

LInstance LC_GenerateAlign(char *classname, char *instname, char **params);

Description

Creates an instance of the T-Cell classname and positions it next to the last placed instance. The T-Cell instance is created using parameter values specified by params, and it is assigned the instance name instname. This function also increments the current (x,y) position according to the current composition direction.

Return Value

Returns a pointer to the newly created instance.

Parameters

- **classname**: Name of the T-Cell to be instanced.
- **instname**: Name of the newly created instance.
- **params**: T-Cell parameter values.

Example

LComp_SetPlacementOrientation(LRotate180MirrorX);
LComp_GenerateAlign("CELL", "NAME", tparams);

See Also

“LC_Generate” (page 1838)
Position Functions

Position functions allow you to retrieve information about the position or size of an element:

“LC_GetPoint” (page 1843)          “LC_GetPointEX” (page 1844)
“LC_GetPlacementRect” (page 1845)  “LC_GetPlacementRectEX” (page 1846)
“LC_GetElementWidth” (page 1847)   “LC_GetElementHeight” (page 1848)
LC_GetPoint

LPoint LC_GetPoint(char *element);

Description

Gets the (x,y) location, in internal units, of the specified element. The location is specified using the current settings for the element boundary and reference point. To get the element’s location with respect to particular boundary and reference point definitions, use “LC_GetPointEX” (page 1844).

Return Value

Returns an (x, y) location in internal units.

Parameters

The parameter element is the name of a cell, instance, or port (e.g., “Topcell/InstA/PortB”). See “Elements” on page 1812 for an explanation of element names.

See Also

“LC_GetPointEX” (page 1844), “LPoint” (page 1780)
LC_GetPointEX

LPoint LC_GetPointEX(AbutTo rect, RelativeTo point, char *element);

Description

Gets the location of the specified element, using the abutment type and reference point specified by rect and point, respectively.

Return Value

Returns the (x, y) position, in internal units, of the specified element.

Parameters

rect

Specifies the abutment type, which determines the boundary definition for the specified element. Must be one of the following values:

- MBB—Minimum bounding box of the cell’s geometry, excluding labels. For ports, MBB is the smallest box that encloses the port and its associated label.
- ABUT—Smallest box that encloses all geometry on the cell’s abutment layer. For ports, ABUT is equal to the port box, excluding its label.

point

Reference point at which the element’s location is reported. Possible values are:

- LEFT—center left point of the bounding box
- RIGHT—center right point of the bounding box
- CENTER—center of the bounding box
- TOP—upper center point of the bounding box
- BOTTOM—lower center point of the bounding box
- UL—upper left corner of the bounding box
- LL—lower left corner of the bounding box
- UR—upper right corner of the bounding box
- LR—lower right corner of the bounding box
- ORIGIN—origin of the instanced cell’s coordinate system

element

Name of a cell, instance, or port (e.g., “Topcell/InstA/PortB”). See “Elements” on page 1812 for an explanation of element names.

See Also

“LC_GetPoint” (page 1843), “LPoint” (page 1780)
LC_GetPlacementRect

```c
LRect LC_GetPlacementRect(char *element);
```

**Description**

Gets the coordinates corresponding to the rectangle that bounds the specified element. The element boundary is defined according to the current abutment type; see “LC_SetAbutmentType” (page 1817).

**Return Value**

Returns two pairs of coordinates representing the upper right and lower left corners of the rectangle. See “LRect” (page 1783) for the structure definition.

**Parameters**

The parameter `element` is the name of a cell, instance, or port (e.g., “Topcell/InstA/PortB”). See “Elements” on page 1812 for an explanation of element names.

**See Also**

LC_GetPlacementRectEX

LRect LC_GetPlacementRectEX(AbutTo recttype, char *element);

Description

Gets the coordinates corresponding to the rectangle that bounds the specified element. The boundary definition used (abutment or minimum bounding box) is specified by the recttype parameter.

Return Value

Returns two pairs of coordinates representing the upper right and lower left corners of the rectangle. See “LRect” (page 1783) for the structure definition.

Parameters

recttype

Specifies the abutment type, which determines the boundary definition for the specified element. Must be one of the following values:

- **MBB**—Minimum bounding box of the cell’s geometry, excluding labels. For ports, MBB is the smallest box that encloses the port and its associated label.
- **ABUT**—Smallest box that encloses all geometry on the cell’s abutment layer. For ports, ABUT is equal to the port box, excluding its label.

element

Name of a cell, instance, or port (e.g., “Topcell/InstA/PortB”). See “Elements” on page 1812 for an explanation of element names.

See Also

**LC_GetElementWidth**

```c
LCoord LC_GetElementWidth(char *element);
```

**Definition**

Gets the width, in internal units, of the default bounding box for the specified element.

**Return Value**

Returns the width in internal units.

**Parameters**

The parameter `element` is the name of a cell, instance, or port (e.g., “Topcell/InstA/PortB”). See “Elements” on page 1812 for an explanation of element names.

**See Also**

“LC_GetElementHeight” (page 1848)
LC_GetElementHeight

```
LCoord LC_GetElementHeight(char *element);
```

**Description**

Gets the height of the specified element’s default bounding box.

**Return Value**

Returns the height in internal units.

**Parameters**

The parameter `element` is the name of a cell, instance, or port (e.g., “Topcell/InstA/PortB”). See “Elements” on page 1812 for an explanation of element names.

**See Also**

“LC_GetElementWidth” (page 1847)
Geometry Functions

The following geometry functions allow you to automate creation of drawn objects in the active cell:

- “LC_StartWire” (page 1850)
- “LC_EndWire” (page 1852)
- “LC_CreateCircle” (page 1854)
- “LC_StartPolygon” (page 1856)
- “LC_EndPolygon” (page 1858)
- “LC_AddWirePoint” (page 1851)
- “LC_CreateBox” (page 1853)
- “LC_CreatePort” (page 1855)
- “LC_AddPolygonPoint” (page 1857)
LC_StartWire

LStatus LC_StartWire(char *layername, LCoord width);

Description

Starts a wire of width width on the specified layername. To add points to the wire, including a start and end point, use “LC_AddWirePoint” (page 1851).

Return Value

LStatusOK if successful. If an error occurs, returns the error value.

Parameters

layername Name of the layer on which to begin drawing a wire (e.g., “Metal1”).
width Width of the wire in internal units.

See Also

“LC_AddWirePoint” (page 1851), “LC_EndWire” (page 1852)
**LC_AddWirePoint**

```c
LPoint LC_AddWirePoint(LPoint point);
```

**Description**

Adds a point at location `point` to a wire created with “LC_StartWire” (page 1850).

**Note:** If a new wire has not been created with LC_StartWire, an error message is displayed.

**Return Value**

Returns the new wire point.

**Parameters**

The parameter `point` is the (x, y) location, in internal units, at which to create the next wire point.

**See Also**

“LC_StartWire” (page 1850), “LC_EndWire” (page 1852), “LPoint” (page 1780)
**LC_EndWire**

```c
LObject LC_EndWire(void);
```

**Description**

Ends the wire in progress, which was created with "LC_StartWire" (page 1850).

**Note:**

If a new wire has not been created with LC_StartWire, an error message is displayed.

**Return Value**

Returns a pointer to the completed wire.

**See Also**

"LC_StartWire" (page 1850), "LC_AddWirePoint" (page 1851), "LObject" (page 1769)
LC_CreateBox

LObject LC_CreateBox(char *layername, LPoint lowerleft, LPoint upperright);

Description

Draws a box on the layer named `layername`, with the lower left and upper right corners located at the points `lowerleft` and `upperright`, respectively.

Return Value

Returns a pointer to the box.

Parameters

- `layername` Name of the layer on which to draw the box (e.g., “Poly”).
- `lowerleft` Point specifying the (x,y) location, in internal units, of the lower left corner of the box.
- `upperright` Point specifying the (x,y) location, in internal units, of the upper right corner of the box.

See Also

“LPoint” (page 1780), “LObject” (page 1769)
LC_CreateCircle

```c
LObject LC_CreateCircle(char *layername, LPoint center, LCoord radius);
```

**Description**

Draws a circle on the layer named `layername`, with the specified `center` point and `radius`.

**Return Value**

Returns a pointer to the newly created circle.

**Parameters**

- `layername` Name of the layer on which to draw the circle.
- `center` (x, y) coordinates, in internal units, of the circle’s center point.
- `radius` Radius, in internal units, of the circle.

**See Also**

“LPoint” (page 1780), “LCoord” (page 1707), “LObject” (page 1769)
LC_CreatePort

LPort LC_CreatePort(char *layername, LPoint lowerleft, LPoint upperright, char *portname);

Description

Creates a port named `portname` on layer `layername`. The lower left and upper right corners of the port box are located at the points `lowerleft` and `upperright`, respectively.

Return Value

Returns a pointer to the newly created port.

Parameters

- `layername` Layer on which to create the port.
- `lowerleft` $(x, y)$ coordinates, in internal units, of the lower left corner of the port box.
- `upperright` $(x, y)$ coordinates, in internal units, of the upper right corner of the port box.
- `portname` Name of the newly created port.

See Also

“LPoint” (page 1780), “LObject” (page 1769)
**LC_StartPolygon**

```c
LStatus LC_StartPolygon(char *layername);
```

**Description**

Starts drawing a polygon on the specified layer.

**Return Value**

LStatusOK if successful. If an error occurs, returns the error value.

**Parameters**

The parameter `layername` is the name of the layer on which to begin drawing a polygon.

**See Also**

“LC_AddPolygonPoint” (page 1857), “LC_EndPolygon” (page 1858), “LStatus” (page 1792)
**LC_AddPolygonPoint**

```c
LPoint LC_AddPolygonPoint(LPoint point);
```

**Description**

Adds a point at the location `point` to the current polygon, which is created with “LC_StartPolygon” (page 1856).

**Note:**
If a new polygon has not been created with LC_StartPolygon, an error message is displayed.

**Return Value**

Returns the new polygon point.

**Parameters**

The parameter `point` is the (x,y) location, in internal units, at which to create the next polygon point.

**See Also**

“LC_StartPolygon” (page 1856), “LC_EndPolygon” (page 1858), “LPoint” (page 1780)
LC_EndPolygon

LObject LC_EndPolygon(void);

Description

Ends the polygon in progress, which was created with “LC_StartPolygon” (page 1856).

Note:
If a new polygon has not been created with LC_StartPolygon, an error message is displayed.

Return Value

Returns a pointer to the newly created polygon.

See Also

“LC_StartPolygon” (page 1856), “LC_AddPolygonPoint” (page 1857), “LObject” (page 1769)
Utility Functions

Utility functions include file and cell management, conversions between technology units, and “push” and “pop” functions for saving or retrieving a previous state.

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>“LC_Push”</td>
<td>1860</td>
</tr>
<tr>
<td>“LC_Pop”</td>
<td>1861</td>
</tr>
<tr>
<td>“LC_DiskFileExists”</td>
<td>1862</td>
</tr>
<tr>
<td>“LC_DiskFileDelete”</td>
<td>1863</td>
</tr>
<tr>
<td>“LC_DiskFileRename”</td>
<td>1864</td>
</tr>
<tr>
<td>“LC_Microns”</td>
<td>1866</td>
</tr>
<tr>
<td>“LC_InMicrons”</td>
<td>1867</td>
</tr>
<tr>
<td>“LC_CellNew”</td>
<td>1869</td>
</tr>
<tr>
<td>“LC_CellClose”</td>
<td>1870</td>
</tr>
<tr>
<td>“LC_CellExists”</td>
<td>1871</td>
</tr>
</tbody>
</table>
**LC_Push**

```c
void LC_Push(void);
```

**Description**

Saves the current state to a stack, where it can be retrieved using “LC_Pop” (page 1861). **LC_Push** holds the following state variables in memory:

- current file and cell
- last placed instance
- (x,y) placement position
- placement overlap
- abutment type
- reference point
- orientation
- abutment layer

**See Also**

“LC_Pop” (page 1861)
LC_Pop

void LC_Pop(void);

Description

Retrieves the last state saved with “LC_Push” (page 1860).

See Also

“LC_Push” (page 1860)
LC_DiskFileExists

    int LC_DiskFileExists(char *name);

Description

Checks to see if the file *name* or *name.tdb* exists.

Return Value

Returns 1 if the file exists, 0 otherwise.

Parameters

The parameter *name* is the filename, in quotes, of the file being checked.

See Also

“LC_CellExists” (page 1871)
LC_DiskFileDelete

LStatus LC_DiskFileDelete(char *filename);

Description

Deletes the file named filename or filename.tdb.

Return Value

LStatusOK if successful. If an error occurs, returns the error value.

Parameters

The parameter name is the filename, in quotes, of the file being deleted.

See Also

**LC_DiskFileRename**

```c
LStatus LC_DiskFileRename(char *from, char *to);
```

**Description**

Renames the specified file to the new name.

**Return Value**

LStatusOK if successful. If an error occurs, returns the error value.

**Parameters**

- `from` Original filename, in quotes.
- `to` New filename, in quotes.

**See Also**

“LC_DiskFileExists” (page 1862), “LC_DiskFileDelete” (page 1863)
LC_Lambda

```
LCoord LC_Lambda(double lambda);
```

Description

Converts an input value from lambda units to internal units.

Return Value

Returns the input value in internal units.

Parameters

The parameter `lambda` is a distance expressed in “lambda” units.

See Also

“LC_Microns” (page 1866), “LC_InMicrons” (page 1867)
LC_Microns

\[ \text{LCoord LC_Microns(double } \text{dist}); \]

Description

Converts an input value from microns to internal units.

Return Value

Returns the input value in internal units.

Parameters

The parameter \text{dist} is a distance expressed in microns.

See Also

“LC_Lambda” (page 1865), “LC_InMicrons” (page 1867)
LC_InMicrons

    double LC_InMicrons(LCoord internal);

Description

Converts an input value from internal units to microns.

Return Value

Returns the input value in microns.

Parameters

The parameter internal is a distance expressed in internal units.

See Also

“LC_Lambda” (page 1865), “LC_Microns” (page 1866)
**LC_CellOpen**

```
LCell LC_CellOpen(char *cellname);
```

**Description**

Opens the cell named `cellname`, or creates a new cell named `cellname`.

**Return Value**

Returns a pointer to the opened cell.

**Parameters**

The parameter `cellname` is the name of the cell to be opened.

**See Also**

**LC_CellNew**

```c
LCell LC_CellNew(char *cellname);
```

**Description**

Creates a new cell named `cellname`. If a cell named `cellname` already exists, its contents are deleted.

**Return Value**

Returns a pointer to the new cell.

**Parameters**

The parameter `cellname` is the name of the new cell.

**See Also**


LC_CellClose

LStatus LC_CellClose(int flags);

/* flags for CellClose */
#define UPDATE_ABUT   1
#define LOCK_CELL    2
#define UNLOCK_CELL  4
#define NO_LIST      8

Description

Closes the current cell and applies the specified flags.

Return Value

LStatusOK if successful. If an error occurs, returns the error value.

Parameters

Any combination of the following flags may be used:

- **UPDATE_ABUT**: Updates the cell’s abutment box on the abutment layer.
- **LOCK_CELL**: Locks the cell to prevent editing.
- **UNLOCK_CELL**: Unlocks the cell to allow editing.
- **NO_LIST**: Hides the cell from the cell list in the Design Navigator and other cell dialogs.

See Also

LC_CellExists

    int LC_CellExists(char *cellname);

Description

Checks to see if the cell named cellname exists in the current TDB file.

Return Value

Returns 1 if the cell exists, 0 otherwise.

Parameters

The parameter cellname is the name of the cell to be checked.

See Also

LC_PropagatePorts

LStatus LC_PropagatePorts(char *instname, char *suffix);

Description

Creates duplicates of all ports in the instance named instname and places them in the current cell. The duplicate ports are named PORTNAMEsuffix, where PORTNAME is the name of the original port.

Return Value

LStatusOK if successful. If an error occurs, returns the error value.

Parameters

- **instname**: Name of the instance, in quotes, from which to duplicate ports.
- **suffix**: Text string that will be appended to the names of new (duplicate) ports (e.g., “.bottom”). If the suffix is NULL, port names are unchanged from the originals.
LC_Trace

```c
void LC_Trace(int trace);
```

**Description**

Turns tracing on or off. When tracing is on, every LComp function reports its execution history to the UPI log file. To specify a separate file for trace output, use “LC_TraceFile” (page 1874).

**Parameters**

- **trace**
  
  On or off state of LComp tracing. *Only* two values are supported:
  - **ON**—turns tracing on
  - **OFF**—turns tracing off

**See Also**

“LC_TraceFile” (page 1874)


**LC_TraceFile**

```c
void LC_TraceFile(char *filename);
```

**Description**

Specifies a file for trace output. When tracing is on, every LComp function traces its execution history to the trace file `filename`. Use “**LC_Trace**” (page 1873) to turn tracing on and off.

**Parameters**

- `filename` Name of the file for trace output; the filename must be enclosed in quotes.

**See Also**

“**LC_Trace**” (page 1873)
LC_PlaceMarkerAtCurrentPos

LPoint LC_PlaceMarkerAtCurrentPos(char *label);

Description

Places a point port with the text \textit{label} at the current \((x, y)\) position. Point ports can be useful for debugging.

Return Value

Returns a pointer to the current \((x, y)\) position.

Parameters

\begin{itemize}
  \item \textit{label} \hspace{1cm} Text string to label the newly created point port (e.g., \texttt{"test\_port"}).
\end{itemize}
Typedefs

“AbutTo” (page 1877)
“CompositionDirectionType” (page 1878)
“RelativeTo” (page 1879)
AbutTo

typedef enum
{
    MBB, Mbb = MBB,
    ABUT, Abut = ABUT
} AbutTo;

Description

**MBB**

Defines the bounding box of a cell as the smallest box that encloses all geometry in the cell, ignoring port labels. For a port, the minimum bounding box is the smallest box that encloses all of the port, excluding its label.

**ABUT**

Defines the bounding box of a cell as the smallest box that encloses all geometry on the abutment layer, which is typically the Icon layer in L-Edit. Specify the abutment layer using `LC_SetAbutmentLayer`. For ports, the port box (excluding the label) is the bounding box.
CompositionDirectionType

typedef long CompositionDirectionType;

Definition

Direction in which instances are placed consecutively. Only the following values are supported:

- UP (or VERTICAL)
- DOWN
- LEFT
- RIGHT (or HORIZONTAL)
- NONE

When set to NONE, the \((x, y)\) placement position is not incremented after placing an instance.
**RelativeTo**

```c
typedef long RelativeTo;
```

**Description**

Reference point for LComp placement and position functions. The reference point can be the origin of the source cell’s coordinate system, or one of nine points on the object’s bounding box:

When the value **ORIGIN** is used, the reference point is the origin of the source (instanced) cell’s coordinate system.
Examples

The following pages show the T-Cell code used in four L-Edit design examples:

- “Logo Generator” (page 1881)
- “Buffer” (page 1883)
- “Matched Dual Capacitor Array” (page 1885)
- “Decoder” (page 1892)

All four examples are included as TDB sample files with L-Edit. They are located in the folder `install_dir/Samples/T-Cells`, where `install_dir` is the directory in which L-Edit was installed.
Logo Generator

The logo generator is included in the file *install_dir* /samples/T-Cells/LayoutText.tdb. This file includes a set of layout cells that each represent a single character. The T-Cell **LOGO** converts a user-specified text string to block letters using alphabet cells.

For tdb designs with libraries, the "Library Name" should be set to the toplevel library and disabled, as is done for "Group," "New Cell," and other commands that create cells.

See Also “**LCell_MakeLogo**” (page 1279).

**T-Cell Code**

```c
module LOGO_code
{
    //include <stdlib.h>
    #include <math.h>
    #include <string.h>
    #include <stdio.h>
    #include "ldata.h"
    #include "lcomp.h"

    void PlaceCharacter(char ch)
    {
        if ( isalnum( ch ) )
            LC_Position( LFormat("_alphabet_%c", toupper( ch )));
        /* otherwise, need to special-case it */
        else if (isspace( ch ))
            LC_Position("_alphabet_space");
        else {
            switch ( ch ) {
                case '+': LC_Position("_alphabet_+"); break;
                case '-': LC_Position("_alphabet_-" ); break;
                case '_': LC_Position("_alphabet_0"); break;
                case '*': LC_Position("_alphabet_*"); break;
                case ':': LC_Position("_alphabet_colon"); break;
                case '\': LC_Position("_alphabet_apostrophe"); break;
                case '/': LC_Position("_alphabet_slash"); break;
                case '\\': LC_Position("_alphabet_backslash"); break;
                case '.': LC_Position("_alphabet_period"); break;
                case ',': LC_Position("_alphabet_comma"); break;
                case '(' : LC_Position("_alphabet_(") ; break;
                case ')': LC_Position("_alphabet_") ; break;
                default:
                    LDialog_MsgBox ( LFormat ( "Unknown character '%c'
                        (code %d)
                        
                { LData\n                    , ch , ch }));
                
            }
        }
    }

    void convert_string(char *s)
    {
        while (*s)
        {
            /* note hardwired check for \ */
            if (*s == 92 && *(s+1) == 'n')
            {
```
```c
char *f = s;
*f++ = '\n';
for (; *(f+1) ; f++)
    *f = *(f+1);
*f = '\0';
s++;
}

/* TODO: Put local variables and functions here. */
void LOGO_main(void)
{
    /* Parameter variables */
    LCell cellCurrent;
    const char* text;
    char s[1000];
    char *t;
    int first = 1;
    /* Initialize parameter variables */
    cellCurrent = (LCell)LMacro_GetNewTCell();
    text = (const char*)LCell_GetParameter(cellCurrent, "text");
    /* TODO: Begin custom generator code. */
    strcpy(s, text);
    convert_string(s);
    LC_InitializeState();
    LC_CurrentCell = cellCurrent;
    LC_SetCompositionDirection(HORIZONTAL);
    LC_SetReferencePoint(LL);
    for (t = s; *t; t++)
    {
        if ( *t == '\n' )
        {
            LC_SetXPlacementPosition(0);
            LC_IncrementYPlacementPosition(-LC_GetElementHeight("_alphabet_A"));
        }
        else {
            PlaceCharacter(*t);
        }
    }
    LC_CellClose(UPDATE_ABUT);
    /* End custom generator code. */
    }
LOGO_main();
```
Buffer

The file `install_dir/samples/T-Cells/buffer.tdb` illustrates a buffer generator. The T-Cell buffer contains code to generate a buffer cell with a user-specified drive value.

T-Cell Code

```c
module BUFFER_code
{
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include "ldata.h"

/* Begin -- Remove this block if you are not using L-Comp. */
#include "lcomp.h"
/* End */

/* TODO: Put local variables and functions here. */
void BUFFER_main(void)
{
  /* Parameter variables */
  LCell cellCurrent;
  int DRIVE;
  int in, out;
  LPoint pt;

  /* Initialize parameter variables */
  cellCurrent = (LCell)LMacro_GetNewTCell();
  DRIVE = (int)LCell_GetParameter(cellCurrent, "DRIVE");

  /* Begin -- Remove this block if you not using L-Comp. */
  LC_InitializeState();
  LC_CurrentCell = cellCurrent;
  /* End */

  /* TODO: Begin custom generator code.*/

  /* check for sane argument */
  if (! ( DRIVE > 0 && DRIVE <= 100 ) )
    {
      LDialog_MsgBox("Buffer drive must be between 1 and 100");
      LUpi_SetReturnCode(1); /* error */
      return;
    }

  LC_SetAbutmentType(ABUT);
  LC_SetCompositionDirection(RIGHT);

  /* put down a port used for our final Abut box */
  pt.x = -LC_Lambda(4.5);
  pt.y = 0;
  LC_CreatePort( LC_GetAbutmentLayer(), pt, pt, "Lower left");

  /* first, figure out how many first-stage drivers we need */
```
in = 1 + ( DRIVE - 2 ) / 3;

/* now, build the first-stage */
for ( ; in > 0; in-- )
{
    if ( in & 1 ) LC_SetPlacementOrientation(NONE);
    else LC_SetPlacementOrientation(LRotate0MirrorX);
    LC_InstanceAlign("BUFFER_SLICE", LFormat("in_%d", in));
}

/* put down the core */
LC_InstanceAlign("BUFFER_CORE", "core");

/* build the output stage */
for (out = 0; out < DRIVE; out++ )
{
    if ( out & 1 )
        LC_SetPlacementOrientation(LRotate0MirrorX);
    else LC_SetPlacementOrientation(NONE);
    LC_InstanceAlign("BUFFER_SLICE", LFormat("out_%d", out));
}

/* propagate the I/O ports */
LC_PropagatePorts("core", "");

/* now update the Abutment box */
/* put down a port used for our final Abut box */
pt.x = LC_x + LC_Lambda(4.5);
pt.y = 0;
LC_CreatePort( LC_GetAbutmentLayer(), pt, pt, "Lower right");

LC_CellClose(UPDATE_ABUT);

/* End custom generator code.*/

BUFFER_main();
Matched Dual Capacitor Array

The file for this example is `install_dir/samples/T-Cells/MatchedDualCapacitorArray.tdb`. The T-Cell `MatchedDualCapacitorArray` creates two matched capacitors out of an array of unit capacitors. It also creates dummy capacitors around the array to improve matching.

**T-Cell Code**

```c
module capacitorarray_code
{
  #include <stdlib.h>
  #include <math.h>
  #include <string.h>
  #include <stdio.h>
  #include "ldata.h"

  ifdef LCOMP
  #include "lcomp.h"
  endif

  ******************************************************************************
  //* GLOBAL Variables:
  ******************************************************************************
  LCoord m1width, m2width, actwidth, polywidth, cntwidth, actovercnt,
  polyovercnt, m1overcnt, m2overvia, seloveract;

  /******************************************************************************
  //* FUNCTIONS:
  ******************************************************************************

  void square( char * layer, LPoint center, LCoord width )
  {
    LPoint pt1, pt2;
    pt1.x = center.x - width / 2;
    pt1.y = center.y - width / 2;
    pt2.x = center.x + width / 2;
    pt2.y = center.y + width / 2;
    LC_CreateBox( layer, pt1, pt2);
  }

  void cell(LPoint center, int flip, LCoord sz)
  {
    LPoint pt = center;
  }

  void capacitorarray_main(void)
  {
    /* Parameter variables */
    LCell cellCurrent;
    int x;
    int y;
    double size;

    /* Initialize parameter variables */
    cellCurrent = (LCell)LMacro_GetNewTCell();
    x = (int)LCell_GetParameter(cellCurrent, "x");
```
Chapter 35: LComp Functions Reference

Chapter 35: LComp Functions Reference

y = (int)LCell_GetParameter(cellCurrent, "y");
size = atof((const char*)LCell_GetParameter(cellCurrent, "size"));

/* Begin -- Remove this block if you not using L-Comp. */
LC_InitializeState();
LC_CurrentCell = cellCurrent;
/* End */

/* TODO: Begin custom generator code.*/
{
    LCoord sz;
    int i, j;
    LPoint pt;
    LCoord xpitch, ypitch;
    LCoord xmin, xmax, ymin, ymax;
    LPoint newpt;

    if(size < 16)
    {
        LDialog_MsgBox("Can't build T-Cell cap_cell with
dimension less than 16");
        LUpi_SetReturnCode(1);
        return;
    }

    //Warn if there are an odd number of unit capacitors.
    if((x*y) % 2)
    {
        if(LDialog_YesNoBox("There are an odd number of
unit capacitors.\nThis will causes the a mismatch
between the two capacitors.\nDo you want to
continue?") == LCANCEL)
        {
            LUpi_SetReturnCode(1);
            return;
        }
    }

    /* initialize all the variables used for design rule
dimensions */
    sz = LC_Microns(size);

    m1width = LC_Microns(3);
    m2width = LC_Microns(3);
    actwidth = LC_Microns(3);
    polywidth = LC_Microns(2);
    cntwidth = LC_Microns(2);
    actovercnt = LC_Microns(1.5);
    polyovercnt = LC_Microns(1.5);
    m1overcnt = LC_Microns(1);
    seloveract = LC_Microns(2);
    m2overvia = LC_Microns(1);

    xpitch = sz + 2*polywidth + 2*polyovercnt + 2*cntwidth +
polyovercnt;
    ypitch = sz + 2*polywidth + 2*actovercnt + cntwidth +
    2*actwidth;

    xmin = xmax = ymin = ymax = 0;
for(j = 0; j < y+2; j++)
{
    pt.y = j * ypitch;
    for (i = 0; i < x+2; i++)
    {
        pt.x = i * xpitch;
        /* now actually build the capacitor */
        square("Poly", pt, sz);
        square("Capacitor ID", pt, sz);
        // Don't forget to add Cap ID to the
        // section of poly
        // that routes to the cap.
        {
            LPoint pt1, pt2;
            if(j & 1)
            {
                pt1.x = pt.x - sz/2 -
                        polywidth;
                pt2.x = pt.x - sz/2;
            }
            else
            {
                pt1.x = pt.x + sz/2;
                pt2.x = pt.x + sz/2 +
                        polywidth;
            }
            LCoord nPolyRoutingWidth = (cntwidth
                                        + 2*polyovercnt);
            pt1.y = pt.y - (nPolyRoutingWidth/
                           2);
            pt2.y = pt1.y + nPolyRoutingWidth;
            LC_CreateBox("Capacitor ID", pt1,
                          pt2);
        }
        square("Active", pt, sz + 2*polywidth);
        // Mark the caps that are dummy caps for
        // matching.
        if((j==0) || (j==y+1) || (i==0) ||
           (i==x+1))
        {
            LPoint pt1 = LPoint_Set(pt.x-sz/2,
                        pt.y-sz/2);
            LPoint pt2 = LPoint_Set(pt1.x+sz,
                        pt1.y+sz);
            LPort pPort = LC_CreatePort("Label",
                        pt1, pt2, "Dummy");
            LPort_SetTextAlignment(pPort,
                        PORT_TEXT_MIDDLE|PORT_TEXT_CENTER);
        }
    }
}
/* poly connection */
newpt = pt;
if (j & 1)
    newpt.x -= xpitch/2;
else
    newpt.x += xpitch/2;
square("Poly", newpt, cntwidth + 2*polyovercnt);

LC_CreatePort("Metal1", newpt, newpt, LFormat("A[%d][%d]", i, j));
if ( i > 0 && i <= x && j > 0 && j <= y )
{
    square("Poly Contact", newpt, cntwidth);
    square("Metal1", newpt, cntwidth + 2*m1overcnt);
    square("Via1", newpt, cntwidth);
    square("Metal2", newpt, cntwidth + m2overvia);
}

LC_StartWire("Poly", cntwidth + 2*polyovercnt);
LC_AddWirePoint(newpt);
LC_AddWirePoint(pt);
LC_EndWire();
if ( newpt.x < xmin ) xmin = newpt.x;
if ( newpt.x > xmax ) xmax = newpt.x;

/* active connection */
newpt = pt;
if (i & 1)
    newpt.y -= ypitch/2;
else
    newpt.y += ypitch/2;
square("Active", newpt, cntwidth + 2*actovercnt);
if ( i > 0 && i <= x && j > 0 && j <= y )
{
    square("Active Contact", newpt, cntwidth);
    square("Metal1", newpt, cntwidth + 2*mlovercnt);
}

LC_CreatePort("Metal1", newpt, newpt, LFormat("B[%d][%d]", i, j));
LC_StartWire("Active", cntwidth+2*actovercnt);
LC_AddWirePoint(newpt);
LC_AddWirePoint(pt);
LC_EndWire();
if ( newpt.y < ymin ) ymin = newpt.y;
if ( newpt.y > ymax ) ymax = newpt.y;
}
}

/* now it's time to wire up the array */
/* first, vertical wires */
for ( i = 1; i < x+2; i++ )
{
    LPoint a, b;
    a = LC_GetPoint(LFormat("A[%d][%d]", i, 1));
    LC_StartWire("Metal2", m2width);
    b.x = a.x;
    if (i & 1)
    {

b.y = ymin + ypitch;
LC_AddWirePoint(b);
b.y = ymax;
LC_AddWirePoint(b);
}
else {
b.y = ymax - ypitch;
LC_AddWirePoint(b);
b.y = ymin;
LC_AddWirePoint(b);
}
LC_EndWire();
}

/* now, the horizontal wires */
for (j = 1; j < y+2; j++)
{
LPoint a, b;
a = LC_GetPoint(LFormat("B[%d][%d]", 1, j));
LC_StartWire("Metal1", m1width);
b.y = a.y;
if (j & 1)
{
b.x = xmin + xpitch;
LC_AddWirePoint(b);
b.x = xmax;
LC_AddWirePoint(b);
}
else{
b.x = xmax - xpitch;
LC_AddWirePoint(b);
b.x = xmin;
LC_AddWirePoint(b);
}
LC_EndWire();
}

/* wire up the outside */
/* bottom */
newpt.x = 0;
newpt.y = ymin;
LC_StartWire("Metal2", m2width);
LC_AddWirePoint(newpt);
newpt.x = xmax - xpitch/2;
LC_AddWirePoint(newpt);
LC_EndWire();

// Write the capacitance as a port, if possible.
double dUnitCapacitance = 0; // in fF.
LLayer pLayer = LLayer_Find(LC_CurrentFile, "NMOS Capacitor");
if(Assigned(pLayer))
{
LLayerParamEx830 oLayerParameters;
LLayer_GetParametersEx830(pLayer, &oLayerParameters);
if(oLayerParameters.AreaCapacitance > 0)
{
LCoord nPolyRoutingWidth = (cntwidth + 2*polyovercnt);
double dArea = size*size + 
    LC_InMicrons(polywidth)*LC_InMicrons(nPolyRoutingWidth);

dUnitCapacitance +=
oLayerParameters.AreaCapacitance*dArea
    /1000; // Convert to fF.
}

if(oLayerParameters.FringeCapacitance>0)
{
    // We don't add nPolyRoutingWidth too the
    // perimeter because one of the
    // nPolyRoutingWidth sides is shared with
    // the main poly capacitor
    // perimeter and should not be included.
    // The other nPolyRoutingWidth
    // side is already included in the main
    // poly capacitor perimeter.
    (4*MainCapSide - Lrouting) + (2*Lrouting +
        2*Wrouting - Lrouting)

double dPerimeter = 4*size +
    2*LC_InMicrons(polywidth);

dUnitCapacitance +=
oLayerParameters.FringeCapacitance*dPerimeter;
}
}

// The bottom wire connects to (x*y)/2 unit capacitors.
int iBottom = ((x*y)/2);
double dCapacitance = iBottom*dUnitCapacitance;
if(dCapacitance > 0)
{
    LPoint pt1 = LPoint_Set(0, ymin-(m2width/2));
    LPoint pt2 = LPoint_Set(xmax - xpitch/2, 
        ymin+(m2width/2));
    LPort pPort = LC_CreatePort("Label Filled", pt1, 
        pt2, LFormat("Cb = %lG fF", dCapacitance));
    LPort_SetTextAlignment(pPort, 
        PORT_TEXT_LEFT|PORT_TEXT_BOTTOM);
}

/* top */
newpt.y = ymax;
LC_StartWire("Metal2", m2width);
LC_AddWirePoint(newpt);
newpt.x = 0;
LC_AddWirePoint(newpt);
LC_EndWire();

// The top wire connects to (x*y)/2 + 1 unit capacitors
// for odd number arrays.
dCapacitance = ((x*y)-iBottom)*dUnitCapacitance;
if(dCapacitance > 0)
{
    LPoint pt1 = LPoint_Set(0, ymax-(m2width/2));
    LPoint pt2 = LPoint_Set(xmax - xpitch/2, 
        ymax+(m2width/2));
    LPort pPort = LC_CreatePort("Label Filled", pt1, 
        pt2, LFormat("Ct = %lG fF", dCapacitance));
    LPort_SetTextAlignment(pPort, 
        PORT_TEXT_LEFT|PORT_TEXT_TOP);
/* left */
newpt.y = 0;
newpt.x = xmin;
LC_StartWire("Metal1", m1width);
LC_AddWirePoint(newpt);
newpt.y = ymax - ypitch/2;
LC_AddWirePoint(newpt);
LC_EndWire();

/* right */
newpt.x = xmax;
LC_StartWire("Metal1", m1width);
LC_AddWirePoint(newpt);
newpt.y = 0;
LC_AddWirePoint(newpt);
LC_EndWire();

pt.x = xmin - LC_Microns(5);
pt.y = ymin - LC_Microns(5);
newpt.x = xmax + LC_Microns(5);
newpt.y = ymax + LC_Microns(5);
LC_CreateBox("N Select", pt, newpt);

}  
/* End custom generator code.*/
} // End of Function:  capacitorarray_main
} // End of Module:  capacitorarray_code

capacitorarray_main();
Decoder

The decoder generator in `install_dir/samples/T-Cells/decoder.tdb` actually uses three additional parameterized subcells to generate a decoder cell with user-specified pitch, number of outputs, and number of bits.

**Note:** When a T-Cell generates subcells that depend on parameter inputs, these subcells must also be defined as T-Cells. Otherwise, they will be overwritten when the parent T-Cell is generated with new parameters.

T-Cell Code: Decoder Generator

```c
module Decoder_Generator_code
{
    #include <math.h>
    #include "lcomp.h"
    #define odd(a)  (a&1)

    void main(void)
    {
        /* Parameter variables */
        LCell       cellCurrent;
        int         Outputs;
        int         DecoderBits;
        int         Pitch;
        bool        Spacers;
        char*       params[5]; /* array of pointers to character strings */
        LRect       rect;

        /* Initialize parameter variables */
        cellCurrent = (LCell)LMacro_GetNewTCell();
        Outputs = (int)LCell_GetParameter(cellCurrent, "Outputs");
        DecoderBits = (int)LCell_GetParameter(cellCurrent, "DecoderBits");
        Pitch = (int)LCell_GetParameter(cellCurrent, "Pitch");
        Spacers = (bool)LCell_GetParameter(cellCurrent, "Spacers");

        /* Begin custom generator code.*/
        if (DecoderBits < 1 || DecoderBits > 8)
        {
            LDialog_MsgBox("Input bits must be between 1 and 8");
            LUpi_SetReturnCode(1);
            return;
        }
        if (Outputs < 1)
        {
            LDialog_MsgBox("Output bits must be positive integer");
            LUpi_SetReturnCode(1);
            return;
        }
        if (DecoderBits < LC_Bits(Outputs-1))
        {
```

```
LDialog_MsgBox(LFormat("Not enough input bits for specified outputs! (need %d)", LC_Bits(Outputs-1))); LUpi_SetReturnCode(1); return;
}
/* Optional: tracing */
// LC_TraceFile("c:\tanner\decoder.log");
// LC_Trace(2);
LC_InitializeState();
LC_CurrentCell = cellCurrent;
LC_SetReferencePoint( LL );
LC_SetCompositionDirection(Vertical);
LC_SetAbutmentType(ABUT);

/* parameter 1, name and value */
params[0] = "DecoderBits"; params[1] = LFormat("%d", DecoderBits);
/* end parameter list with NULL */
params[2] = NULL;
LC_Generate("decoder bottom", "auto decoder bottom", params);  

/* parameter 2, name and value */
params[2] = "Outputs";
params[3] = LFormat("%d", Outputs);
/* end parameter list with NULL */
params[4] = NULL;
LC_Generate("decoder guts", "auto decoder guts", params);
rect = LC_GetPlacementRect("auto decoder guts");
LC_CreateBox("Metal2", LPoint_Set(rect.x0, rect.y0),
            LPoint_Set(rect.x1, rect.y1));
/* end parameter list with NULL */
params[2] = NULL;
LC_Generate("decoder top", "auto decoder top", params);
/* End custom generator code.*/
}
main();

**T-Cell Code: decoder bottom**

module decoder_bottom_code
{
#include "lcomp.h"
#define odd(a)  (a&1)

 /* Put local variables and functions here. */
 void decoder_bottom_main(void)
 {
 /* Parameter variables */
 LCell       cellCurrent;
 int         DecoderBits;
 int i;

 /* Initialize parameter variables */
cellCurrent = (LCell)LMacro_GetNewTCell();
DecoderBits = (int)LCell_GetParameter(cellCurrent, "DecoderBits");

LC_InitializeState();
LC_CurrentCell = cellCurrent;

/* Begin custom generator code.*/
LC_SetCompositionDirection(Horizontal);
LC_SetAbutmentType(ABUT);
LC_Position("row bottom begin");

for (i = 0; i < DecoderBits; i++)
{
    /* mirror alternating cells */
    if (odd(i))
        LC_SetPlacementOrientation(LRotate0MirrorX);
    else
        LC_SetPlacementOrientation(LNormalOrientation);
    LC_Position("row bottom");
}

LC_SetPlacementOrientation(LNormalOrientation);
LC_Position("row bottom end");
LC_CellClose(UPDATE_ABUT);

/* End custom generator code.*/
}
}
decoder_bottom_main();

T-Cell Code: decoder guts

module decoder_guts_code
{
    #include "lcomp.h"
    #define odd(a)  (a&1)

    /* Put local variables and functions here. */
    void decoder_guts_main(void)
    {
        /* Parameter variables */
        LCell    cellCurrent;
        int      Outputs;
        int      DecoderBits;
        int      Pitch;
        int      i, j;
        long     data;
        LCoord   minbitheight;

        /* Initialize parameter variables */
        cellCurrent = (LCell)LMacro_GetNewTCell();
        Outputs    = (int)LCell_GetParameter(cellCurrent, "Outputs");
        DecoderBits = (int)LCell_GetParameter(cellCurrent, "DecoderBits");
        Pitch      = (int)LCell_GetParameter(cellCurrent, "Pitch");
        LC_InitializeState();
        LC_CurrentCell = cellCurrent;

        /* Begin custom generator code.*/
    }
}
LC_SetAbutmentType(ABUT);

Pitch = LC_Lambda(Pitch);
minbitheight = LC_GetElementHeight("bg bitcell 1");

if (Pitch < minbitheight)
{
    LDialog_MsgBox("Pitch is too small");
    return;
}

for (i = 0; i < Outputs; i++)
{
    LC_SetCompositionDirection(NONE);
    LC_SetXPlacementPosition(0);
    LC_PlaceMarkerAtCurrentPos("ROW START");
    LC_SetPlacementOrientation(LNormalOrientation);
    LC_SetCompositionDirection(Horizontal);
    LC_Instance("row begin", LFormat("begin[%d]", i));

    for (j = 0, data = i; j < DecoderBits; j++, data = data >> 1)
    {
        if (odd(j))
            LC_SetPlacementOrientation(LRotate0MirrorX);
        else
            LC_SetPlacementOrientation(LNormalOrientation);

        if (odd(data))
            LC_Instance("bg bitcell 1",
                        LFormat("b[%d][%d]", j, i));
        else
            LC_Instance("bg bitcell 0",
                        LFormat("b[%d][%d]", j, i));
    }
    LC_SetPlacementOrientation(LNormalOrientation);
    LC_Instance("row end", LFormat("end[%d]", i));
    LC_SetPlacementOrientation(LNormalOrientation);
    LC_Instance("row ended", LFormat("end [%d]", i));
    LC_IncrementYPlacementPosition( Pitch );
    LC_SetXPlacementPosition(0);
}

/* wire it up */
LC_SetReferencePoint(CENTER);
for (j = 0; j < DecoderBits; j++)
{
    int l;

    for (l = 0; l < 4; l++)
    {
        LC_StartWire("Poly", LC_Lambda(2));

        LC_AddWirePoint(LC_GetPoint(LFormat("b[%d][%d]/A[%d]", j, 0, l)));
        LC_AddWirePoint(LC_GetPoint(LFormat("b[%d][%d]/A[%d]", j, Outputs-1, l)));
    }
}
Chapter 35: LComp Functions Reference

T-Cell Code: decoder top

module decoder_top_code
{
    #include "lcomp.h"
    #define odd(a)  (a&1)

    /* Put local variables and functions here. */
    void decoder_top_main(void)
    {
        /* Parameter variables */
        LCell       cellCurrent;
        int         DecoderBits;
        int         j;

        LC_EndWire();
    }
    LC_StartWire("N Well",
            LC_GetElementWidth("b[0][0]/WELL"));
    LC_AddWirePoint(LC_GetPoint(LFormat("b[%d][%d]/WELL", j, 0)));
    LC_AddWirePoint(LC_GetPoint(LFormat("b[%d][%d]/WELL", j, Outputs-1)));
    LC_EndWire();
}
LC_StartWire("Metal1", LC_GetElementWidth("begin[0]/Vdd"));
LC_AddWirePoint(LC_GetPoint(LFormat("begin[%d]/Vdd", 0)));
LC_AddWirePoint(LC_GetPoint(LFormat("begin[%d]/Vdd", Outputs-1)));
LC_EndWire();
LC_StartWire("Metal1", LC_GetElementWidth("end[0]/Gnd"));
LC_AddWirePoint(LC_GetPoint(LFormat("end[%d]/Gnd", 0)));
LC_AddWirePoint(LC_GetPoint(LFormat("end[%d]/Gnd", Outputs-1)));
LC_EndWire();

for (j = 0; j < Outputs - 1; j++ )
{
    LC_StartWire("Poly", LC_Lambda(2));
    LC_AddWirePoint(LC_GetPoint(LFormat("end[%d]/NABL_top", j)));
    LC_AddWirePoint(LC_GetPoint(LFormat("end[%d]/NABL_bot", j+1)));
    LC_EndWire();
}
LC_CellClose(UPDATE_ABUT);
/* End custom generator code.*/

} decoder_guts_main();
/* Initialize parameter variables */
cellCurrent = (LCell)LMacro_GetNewTCell();
DecoderBits = (int)LCell_GetParameter(cellCurrent,
    "DecoderBits");

LC_InitializeState();
LC_CurrentCell = cellCurrent;

/* Begin custom generator code.*/
LC_SetCompositionDirection(Horizontal);
LC_SetAbutmentType(ABUT);
LC_Position("row top begin");
for (j = 0; j < DecoderBits; j++)
{
    if (odd(j))
        LC_SetPlacementOrientation(LRotate0MirrorX);
    else
        LC_SetPlacementOrientation(LNormalOrientation);
    LC_Position("row top");
}
LC_SetPlacementOrientation(LNormalOrientation);
LC_Position("row top end");
LC_CellClose(UPDATE_ABUT);
/* End custom generator code.*/
}
}
dercoder_top_main();
### 36 Alphabetical List of UPI and LComp Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAmbiguousFillType</td>
<td>1695</td>
</tr>
<tr>
<td>LApp.ExitAfterCompletion</td>
<td>1192</td>
</tr>
<tr>
<td>LApp.GetAllowSelectionOnLockedLayers</td>
<td>1174</td>
</tr>
<tr>
<td>LApp.GetCacheInstances</td>
<td>1155</td>
</tr>
<tr>
<td>LApp.GetCacheInstancesSmallerThanNumOfPixels</td>
<td>1156</td>
</tr>
<tr>
<td>LApp.GetFillObjectsDuringDrawing</td>
<td>1157</td>
</tr>
<tr>
<td>LApp.GetFullVersion</td>
<td>1167</td>
</tr>
<tr>
<td>LApp.GetHideInstanceInsidesIfLessThanNumOfPixels</td>
<td>1158</td>
</tr>
<tr>
<td>LApp.GetHideObjectsSmallerThanNumOfPixels</td>
<td>1160</td>
</tr>
<tr>
<td>LApp.GetHideSmallInstanceInsides</td>
<td>1161</td>
</tr>
<tr>
<td>LApp.GetHideSmallObjects</td>
<td>1162</td>
</tr>
<tr>
<td>LApp.GetInterruptableRendering</td>
<td>1163</td>
</tr>
<tr>
<td>LApp.GetRedrawAllWindows</td>
<td>1164</td>
</tr>
<tr>
<td>LApp.GetRenderingUseCPUForColorMixing</td>
<td>1171</td>
</tr>
<tr>
<td>LApp.GetRenderingUseMMX</td>
<td>1172</td>
</tr>
<tr>
<td>LApp.GetRenderingUsePatBltForPatterns</td>
<td>1173</td>
</tr>
<tr>
<td>LApp.GetShowDesignFirstTimeIncrement</td>
<td>1169</td>
</tr>
<tr>
<td>LApp.GetShowDesignNextTimeIncrement</td>
<td>1170</td>
</tr>
<tr>
<td>LApp.GetShowDesignWhileRendering</td>
<td>1168</td>
</tr>
<tr>
<td>LApp.GetVersion</td>
<td>1165</td>
</tr>
<tr>
<td>LApp.GetVersionDateTime</td>
<td>1166</td>
</tr>
<tr>
<td>LApp.SetAllowSelectionOnLockedLayers</td>
<td>1191</td>
</tr>
<tr>
<td>LApp.SetCacheInstances</td>
<td>1175</td>
</tr>
<tr>
<td>LApp.SetCacheInstancesSmallerThanNumOfPixels</td>
<td>1176</td>
</tr>
<tr>
<td>LApp.SetExportMaskDataIgnoreHiddenObjects</td>
<td>1177</td>
</tr>
<tr>
<td>LApp.SetFillObjectsDuringDrawing</td>
<td>1178</td>
</tr>
<tr>
<td>LApp.SetHideInstanceInsidesIfLessThanNumOfPixels</td>
<td>1179</td>
</tr>
<tr>
<td>LApp.SetHideObjectsSmallerThanNumOfPixels</td>
<td>1181</td>
</tr>
<tr>
<td>LApp.SetHideSmallInstanceInsides</td>
<td>1182</td>
</tr>
<tr>
<td>LApp.SetHideSmallObjects</td>
<td>1183</td>
</tr>
<tr>
<td>LApp.SetInterruptableRendering</td>
<td>1184</td>
</tr>
<tr>
<td>LApp.SetRedrawAllWindows</td>
<td>1185</td>
</tr>
<tr>
<td>LApp.SetRenderingUseCPUForColorMixing</td>
<td>1188</td>
</tr>
<tr>
<td>LApp.SetRenderingUseMMX</td>
<td>1189</td>
</tr>
<tr>
<td>LApp.SetRenderingUsePatBltForPatterns</td>
<td>1190</td>
</tr>
<tr>
<td>LApp.SetShowDesignTimeIncrement</td>
<td>1187</td>
</tr>
<tr>
<td>LApp.SetShowDesignWhileRendering</td>
<td>1186</td>
</tr>
<tr>
<td>LArcDirection</td>
<td>1696</td>
</tr>
</tbody>
</table>
LAtOF ................................................................. 1692
LBoolean ......................................................... 1697
LBooleanOperation .......................................... 1698
LBox_GetRect .................................................. 1414
LBox_New ......................................................... 1412
LBox_Set ......................................................... 1413
LC_AddPlacementOrientation ......................... 1821
LC_AddPolygonPoint ......................................... 1857
LC_AddWirePoint .............................................. 1851
LC_Align .......................................................... 1839
LC_CellClose .................................................... 1870
LC_CellExists .................................................. 1871
LC_CellNew ....................................................... 1869
LC_CellOpen ..................................................... 1868
LC_CreateBox .................................................... 1853
LC_CreateCircle ................................................. 1854
LC_CreatePort ................................................... 1855
LC_DiskFileDelete ............................................. 1863
LC_DiskFileExists ............................................ 1862
LC_DiskFileRename ............................................ 1864
LC_EndPolygon .................................................. 1858
LC_EndWire ....................................................... 1852
LC_Generate ..................................................... 1838
LC_GenerateAlign ............................................. 1841
LC_GetAbutmentType .......................................... 1818
LC_GetCompositionDirection ......................... 1824
LC_GetElementHeight ......................................... 1848
LC_GetElementWidth .......................................... 1847
LC_GetPlacementOrientation ......................... 1820
LC_GetPlacementOverlap ................................... 1826
LC_GetPlacementRect ......................................... 1845
LC_GetPlacementRectEX ...................................... 1846
LC_GetPoint ...................................................... 1843
LC_GetPointEX ................................................... 1844
LC_GetReferencePoint ........................................ 1816
LC_GetXPlacementPosition .............................. 1830
LC_GetXYPlacementPosition ............................ 1828
LC_GetYPlacementPosition ............................... 1832
LC_InMicrons ....................................................... 1867
LC_Instance ....................................................... 1837
LC_InstanceAlign ............................................. 1840
LC_Lambda ........................................................ 1865
LC_Microns ......................................................... 1866
LC_PlaceMarkerAtCurrentPos .......................... 1875
LC_Pop ............................................................ 1876
LC_Position ....................................................... 1836
LC_PropagatePorts .................................................. 1872
LC_Push ................................................................. 1860
LC_SetAbutmentType ............................................... 1817
LC_SetCompositionDirection ...................................... 1823
LC_SetPlacementOrientation ...................................... 1819
LC_SetPlacementOverlap ............................................ 1825
LC_SetReferencePoint ............................................... 1815
LC_SetXPlacementPosition .......................................... 1829
LC_SetXYPlacementPosition ......................................... 1827
LC_SetYPlacementPosition ............................................ 1831
LC_StartPolygon ...................................................... 1856
LC_StartWire .......................................................... 1850
LC_SubtractPlacementOrientation ................................ 1822
LC_Trace ................................................................. 1873
LC_TraceFile ........................................................... 1874
LCapType ................................................................. 1699
LCell ................................................................. 1700
LCell ................................................................. 1701
LCell_AddMarker .................................................... 1315
LCell_BooleanOperation ............................................ 1318
LCell_CalcMBB ......................................................... 1314
LCell_ClearGenerateLayers ........................................... 1574
LCell_ClearUndoLists ................................................ 1308
LCell_ClearUserData ................................................... 1291
LCell_Copy ............................................................ 1263
LCell_Delete .......................................................... 1262
LCell_DeleteUserData ................................................ 1294
LCell_Find ............................................................. 1264
LCell_FindEx .......................................................... 1265
LCell_Flatten ........................................................... 1290
LCell_GenerateLayers ................................................. 1571
LCell_GenerateLayers_v10_00 ...................................... 1298
LCell_GenerateLayers_v10_00 ...................................... 1573
LCell_GenerateLayers_v11_10 ...................................... 1300
LCell_GenerateLayers_v11_10 ...................................... 1572
LCell_GenerateLayersEx830 ....................................... 1295
LCell_GenerateLayersEx99 ......................................... 1297
LCell_GetAuthor ...................................................... 1273
LCell_GetBasePoint .................................................. 1323
LCell_GetCreatedTime ............................................... 1283
LCell_GetDRCNumErrors ............................................. 1641
LCell_GetDRCStatus ................................................ 1642
LCell_GetFile .......................................................... 1266
LCell_GetInfoText ..................................................... 1277
LCell_GetLastVisible ............................................... 1096
LCell_GetList .......................................................... 1267
LCell_GetLock .......................................................... 1269
LCell_GetMbb ........................................................... 1288
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCell_GetMbbAll</td>
<td>1289</td>
</tr>
<tr>
<td>LCell_GetModifiedTime</td>
<td>1284</td>
</tr>
<tr>
<td>LCellGetName</td>
<td>1271</td>
</tr>
<tr>
<td>LCellGetNext</td>
<td>1268</td>
</tr>
<tr>
<td>LCellGetOrganization</td>
<td>1275</td>
</tr>
<tr>
<td>LCellGetParameter</td>
<td>1309</td>
</tr>
<tr>
<td>LCellGetParameterAsBoolean</td>
<td>1686</td>
</tr>
<tr>
<td>LCellGetParameterAsCoord</td>
<td>1687</td>
</tr>
<tr>
<td>LCellGetParameterAsCoordOnGrid</td>
<td>1688</td>
</tr>
<tr>
<td>LCellGetParameterAsDouble</td>
<td>1689</td>
</tr>
<tr>
<td>LCellGetParameterAsInt</td>
<td>1690</td>
</tr>
<tr>
<td>LCellGetParameterAsLayer</td>
<td>1691</td>
</tr>
<tr>
<td>LCellGetParameterName</td>
<td>1702</td>
</tr>
<tr>
<td>LCellGetParameterType</td>
<td>1703</td>
</tr>
<tr>
<td>LCellGetShowInLists</td>
<td>1313</td>
</tr>
<tr>
<td>LCellGetTCellPreviousValue</td>
<td>1310</td>
</tr>
<tr>
<td>LCellGetUserData</td>
<td>1292</td>
</tr>
<tr>
<td>LCellGetVersion</td>
<td>1281</td>
</tr>
<tr>
<td>LCellGetView</td>
<td>1286</td>
</tr>
<tr>
<td>LCellGetVisible</td>
<td>1095</td>
</tr>
<tr>
<td>LCellHomeController</td>
<td>1094</td>
</tr>
<tr>
<td>LCellIsChanged</td>
<td>1285</td>
</tr>
<tr>
<td>LCellMakeLogo</td>
<td>1279</td>
</tr>
<tr>
<td>LCellMakeVisible</td>
<td>1097</td>
</tr>
<tr>
<td>LCellMakeVisibleNoRefresh</td>
<td>1098</td>
</tr>
<tr>
<td>LCellNew</td>
<td>1261</td>
</tr>
<tr>
<td>LCellOpenDRCStatistics</td>
<td>1640</td>
</tr>
<tr>
<td>LCellOpenDRCSummary</td>
<td>1639</td>
</tr>
<tr>
<td>LCellOpenExtractStatistics</td>
<td>1648</td>
</tr>
<tr>
<td>LCellOpenExtractSummary</td>
<td>1647</td>
</tr>
<tr>
<td>LCellRemoveAllMarkers</td>
<td>1317</td>
</tr>
<tr>
<td>LCellRemoveMarker</td>
<td>1316</td>
</tr>
<tr>
<td>LCellRunDRC</td>
<td>1302</td>
</tr>
<tr>
<td>LCellRunDRCCommandFile</td>
<td>1307</td>
</tr>
<tr>
<td>LCellRunDRCEx00</td>
<td>1303</td>
</tr>
<tr>
<td>LCellRunDRCEx01</td>
<td>1305</td>
</tr>
<tr>
<td>LCellSetAuthor</td>
<td>1274</td>
</tr>
<tr>
<td>LCellSetBasePoint</td>
<td>1322</td>
</tr>
<tr>
<td>LCellSetChanged</td>
<td>1301</td>
</tr>
<tr>
<td>LCellSetInfoText</td>
<td>1278</td>
</tr>
<tr>
<td>LCellSetLock</td>
<td>1270</td>
</tr>
<tr>
<td>LCellSetName</td>
<td>1272</td>
</tr>
<tr>
<td>LCellSetOrganization</td>
<td>1276</td>
</tr>
<tr>
<td>LCellSetShowInLists</td>
<td>1312</td>
</tr>
<tr>
<td>LCellSetUserData</td>
<td>1293</td>
</tr>
<tr>
<td>LCellSetVersion</td>
<td>1282</td>
</tr>
<tr>
<td>LCellSetView</td>
<td>1287</td>
</tr>
<tr>
<td>LCellSlice</td>
<td>1321</td>
</tr>
</tbody>
</table>
LCell_Slice ................................................................. 1321
LCIFParam .............................................................. 1704
LCircle_GetCenter ..................................................... 1418
LCircle_GetRadius ..................................................... 1419
LCircle_GetRect ....................................................... 1420
LCircle_New ............................................................. 1416
LCircle_Set ............................................................. 1417
LColor ................................................................. 1706
LCoord ................................................................. 1707
LCore ................................................................. 1708
LCore_GetCore ......................................................... 1657
LCore_GetLLHCap ...................................................... 1658
LCore_GetLLVCap ...................................................... 1660
LCore_SetLLHCap ...................................................... 1659
LCore_SetLLVCap ...................................................... 1661
LCSV_Run ............................................................... 1684
LCursor_GetPosition ................................................... 1089
LCursor_GetPositionEx99 ............................................. 1090
LCursor_GetSnappedPosition ......................................... 1091
LCursorType .......................................................... 1709
LCurve ................................................................. 1710
LDerivationType ....................................................... 1717
LDerivedLayerAreaOperation ................................-------- 1711
LDerivedLayerBoolOperation ........................................ 1713
LDerivedLayerDensityOperation .................................... 1715
LDerivedLayerParam .................................................. 1716
LDerivedLayerParam .................................................. 1716
LDerivedLayerParamEx830 ............................................ 1718
LDerivedLayerSelectOperation ...................................... 1719
LDesignRuleFlags ..................................................... 1721
LDesignRuleParam ..................................................... 1722
LDialog_AlertBox ...................................................... 1079
LDialog_File .......................................................... 1084
LDialog_InputBox ..................................................... 1081
LDialog_MsgBox ......................................................... 1077
LDialog_MultiLineInputBox .......................................... 1082
LDialog_MultiLineMsgBox ............................................ 1078
LDialog_PickList ..................................................... 1083
LDialog_YesNoBox ..................................................... 1080
LDialogItem ........................................................... 1723
LDisplay_Refresh ...................................................... 1092
LDisplayUnitInfo ...................................................... 1724
LDRC_Run ............................................................... 1634
LDrcFlags ............................................................... 1725
LDrcRule ............................................................... 1726
LDrcRule_Add .......................................................... 1623
LDrcRule_Delete ........................................................ 1624
LDrcRule_DestroyParameter .......................................... 1632
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDrcRule_Find</td>
<td>1625</td>
</tr>
<tr>
<td>LDrcRule_GetList</td>
<td>1626</td>
</tr>
<tr>
<td>LDrcRule_GetNext</td>
<td>1627</td>
</tr>
<tr>
<td>LDrcRule_GetParameters</td>
<td>1630</td>
</tr>
<tr>
<td>LDrcRule_SetParameters</td>
<td>1631</td>
</tr>
<tr>
<td>LDrcRule_SetRuleSet</td>
<td>1628</td>
</tr>
<tr>
<td>LDrcRule_SetTolerance</td>
<td>1629</td>
</tr>
<tr>
<td>LDrcRuleType</td>
<td>1727</td>
</tr>
<tr>
<td>LDrcStatus</td>
<td>1728</td>
</tr>
<tr>
<td>LEntity</td>
<td>1729</td>
</tr>
<tr>
<td>LEntity_AssignBlobProperty</td>
<td>1351</td>
</tr>
<tr>
<td>LEntity_AssignProperty</td>
<td>1350</td>
</tr>
<tr>
<td>LEntity_BrowseProperties</td>
<td>1358</td>
</tr>
<tr>
<td>LEntity_CopyAllProperties</td>
<td>1354</td>
</tr>
<tr>
<td>LEntity_DecompressBlobToFile</td>
<td>1370</td>
</tr>
<tr>
<td>LEntity_DeleteAllProperties</td>
<td>1353</td>
</tr>
<tr>
<td>LEntity_DeleteProperty</td>
<td>1352</td>
</tr>
<tr>
<td>LEntity_GetFirstProperty</td>
<td>1355</td>
</tr>
<tr>
<td>LEntity_GetNextProperty</td>
<td>1356</td>
</tr>
<tr>
<td>LEntity_GetPropertyType</td>
<td>1347</td>
</tr>
<tr>
<td>LEntity_GetPropertyValue</td>
<td>1349</td>
</tr>
<tr>
<td>LEntity_GetPropertyValueSize</td>
<td>1348</td>
</tr>
<tr>
<td>LEntity_LoadBlobProperty</td>
<td>1359</td>
</tr>
<tr>
<td>LEntity_PropertyExists</td>
<td>1346</td>
</tr>
<tr>
<td>LEntity_ReadPropertiesFromFile</td>
<td>1361</td>
</tr>
<tr>
<td>LEntity_SaveBlobProperty</td>
<td>1360</td>
</tr>
<tr>
<td>LEntity_SetCurrentProperty</td>
<td>1357</td>
</tr>
<tr>
<td>LEntity_StoreAsCompressedBlob</td>
<td>1369</td>
</tr>
<tr>
<td>LEntity_StringToValidPropertyName</td>
<td>1363</td>
</tr>
<tr>
<td>LEntity_ValidPropertyNameToString</td>
<td>1365</td>
</tr>
<tr>
<td>LEntity_WritePropertiesToFile</td>
<td>1367</td>
</tr>
<tr>
<td>LExtract_GetOptions_Ex840</td>
<td>1652</td>
</tr>
<tr>
<td>LExtract_GetOptions_Ex98</td>
<td>1651</td>
</tr>
<tr>
<td>LExtract_Run</td>
<td>1644</td>
</tr>
<tr>
<td>LExtract_Run_Dialog</td>
<td>1645</td>
</tr>
<tr>
<td>LExtract_RunCommandFile</td>
<td>1646</td>
</tr>
<tr>
<td>LExtract_RunEx840</td>
<td>1649</td>
</tr>
<tr>
<td>LExtract_SetOptionsEx840</td>
<td>1654</td>
</tr>
<tr>
<td>LExtractOptions</td>
<td>1731</td>
</tr>
<tr>
<td>LExtractOptionsEx840</td>
<td>1733</td>
</tr>
<tr>
<td>LFile</td>
<td>1736</td>
</tr>
<tr>
<td>LFile_ClearUserData</td>
<td>1239</td>
</tr>
<tr>
<td>LFile_Close</td>
<td>1200</td>
</tr>
<tr>
<td>LFile_DefineLayerPalette</td>
<td>1258</td>
</tr>
<tr>
<td>LFile_DeleteUserData</td>
<td>1238</td>
</tr>
<tr>
<td>LFile_DisplayCellBrowser</td>
<td>1240</td>
</tr>
<tr>
<td>LFile_DispUtoIntU</td>
<td>1252</td>
</tr>
</tbody>
</table>
LFile_ExportGDSII .................................................. 1618
LFile_ExportGDSII .................................................. 1620
LFile_ExportGDSII .................................................. 1621
LFile_Find .......................................................... 1201
LFile_GetAuthor ...................................................... 1209
LFile_GetBinSize .................................................... 1635
LFile_GetCIFParameters ............................................ 1610
LFile_GetColorPalette ............................................. 1602
LFile_GetColorPaletteNumColors .................................. 1603
LFile_GetColorPaletteSortBy ...................................... 1604
LFile_GetCurrentLayerPalette ..................................... 1255
LFile_GetCurveSetup ............................................... 1231
LFile_GetDesignRuleFlags ......................................... 1242
LFile_GetDisplayUnitInfo .......................................... 1249
LFile_GetDrcFlags .................................................. 1637
LFile_GetEnvironment .............................................. 1221
LFile_GetFabricationCell ......................................... 1211
LFile_GetGDSParameters ........................................... 1614
LFile_GetGrid ....................................................... 1223
LFile_GetGrid_v10_00 ............................................. 1225
LFile_GetGridEx840 ............................................... 1224
LFile_GetInfoText .................................................. 1219
LFile_GetLayoutVersion .......................................... 1215
LFile_GetList ....................................................... 1202
LFile_GetLock ....................................................... 1204
LFileGetName ........................................................ 1208
LFile_GetNext ....................................................... 1203
LFile_GetNextLayerPalette ........................................ 1257
LFile_GetOrganization .............................................. 1213
LFile_GetResolvedFileName ....................................... 1244
LFile_GetSelectionParam .......................................... 1234
LFile_GetSetupVersion ............................................ 1217
LFile_GetTechnology ............................................... 1591
LFile_GetTechnologyEx840 ........................................ 1599
LFile_GetTopLevelLibraryName .................................... 1259
LFile_GetUserData .................................................. 1236
LFile_GetVisible .................................................... 1246
LFile_ImportCIF .................................................... 1612
LFile_ImportGDSII .................................................. 1617
LFile_IntUtoDispU .................................................. 1251
LFile_IntUtoLocU .................................................... 1247
LFile_IntUtoMicrons ............................................... 1253
LFile_IsChanged ..................................................... 1206
LFile_LocUtoIntU .................................................... 1248
LFile_MicronsToIntU ............................................... 1254
LFile_New ............................................................ 1195
LFile_Open ........................................................... 1196
LFile_OpenCell ....................................................... 1197
Chapter 36: Alphabetical List of UPI and LComp Functions

LFile_Save .................................................. 1198
LFile_SaveAs ............................................... 1199
LFile_SetAuthor ........................................... 1210
LFile_SetBinSize .......................................... 1636
LFile_SetChanged ........................................... 1207
LFile_SetCLIFParameters ................................. 1611
LFile_SetColorPalette ..................................... 1605
LFile_SetColorPaletteNumColors ....................... 1606
LFile_SetColorPaletteSortBy ............................ 1607
LFile_SetCurrentLayerPalette ......................... 1256
LFile_SetCurveSetup ...................................... 1232
LFile_SetDesignRuleFlags ................................ 1243
LFile_SetDisplayUnit ..................................... 1250
LFile_SetDrcFlags ........................................... 1638
LFile_SetEnvironment ..................................... 1222
LFile_SetFabricationCell ................................. 1212
LFile_SetGDSParameters .................................. 1616
LFile_SetGDSParameters .................................. 1618
LFile_SetGDSParameters .................................. 1620
LFile_SetGDSParameters .................................. 1621
LFile_SetGrid ............................................... 1228
LFile_SetGrid_v10_00 .................................... 1226
LFile_SetGridEx840 ....................................... 1229
LFile_SetInfoText ......................................... 1220
LFile_SetLastCurrent ...................................... 1241
LFile_SetLayoutVersion .................................. 1216
LFile_SetLock ............................................. 1205
LFile_SetOrganization .................................... 1214
LFile_SetSelectionParam .................................. 1235
LFile_SetSetupVersion .................................... 1218
LFile_SetTechnology ...................................... 1592
LFile_SetTechnologyEx840 ............................... 1600
LFile_SetTechnologyLambdaDenom ....................... 1598
LFile_SetTechnologyLambdaNum ......................... 1597
LFile_SetTechnologyName .................................. 1593
LFile_SetTechnologyUnitDenom ......................... 1595
LFile_SetTechnologyUnitName ............................ 1596
LFile_SetTechnologyUnitNum ............................. 1594
LFile_SetUserData .......................................... 1237
LFileType .................................................... 1737
LFormat .................................................... 1126
LFormatV ..................................................... 1127
LFormatV ..................................................... 1127
LGDSExport Scope .......................................... 1741
LGDSExportLogParams ...................................... 1745
LGDSParam .................................................... 1738
LGDSParamEx .................................................. 1739
LGeomType ................................................... 1746
Chapter 36: Alphabetical List of UPI and LComp Functions

LGrid ......................................................... 1747
LGrid_v10_00 .............................................. 1750
LGridEx840 ................................................. 1748
LInstance ................................................... 1752
LInstance_Delete ................................. 1327
LInstance_Find ................................. 1330
LInstance_FindNext ............................ 1331
LInstance_Generate ......................... 1342
LInstance_GenerateV ....................... 1343
LInstance_GetCell ............................ 1336
LInstance_GetDelta ......................... 1340
LInstance_GetList ............................ 1332
LInstance_GetMbb .............................. 1341
LInstance_GetName ......................... 1334
LInstance_GetNext ............................. 1333
LInstance_GetRepeatCount .................... 1339
LInstance_GetTransform ...................... 1337
LInstance_GetTransform_Ex99 ............... 1338
LInstance_New ......................................... 1325
LInstance_New_Ex99 ......................... 1326
LInstance_Set ....................................... 1328
LInstance_Set_Ex99 ......................... 1329
LInstance_SetName ......................... 1335
LJoinType ............................................. 1753
LLayer ..................................................... 1755
LLayer_Copy .......................................... 1557
LLayer_Delete ....................................... 1531
LLayer_DestroyDerivedParameter ............. 1568
LLayer_DestroyDerivedParameterEx840 .... 1570
LLayer_DisableAllDerived ................. 1563
LLayer_EnableAllDerived ..................... 1562
LLayer_Find ............................................ 1532
LLayer_FindGDS ..................................... 1533
LLayer_FindLayerName ....................... 1492
LLayer_FindPurposeName .................... 1493
LLayer_GetCap ....................................... 1548
LLayer_GetCurrent ............................. 1552
LLayer_GetDerivedList ....................... 1559
LLayer_GetDerivedNext ...................... 1560
LLayer_GetDerivedParameters ............. 1564
LLayer_GetDerivedParametersEx830 ...... 1565
LLayer_GetList ..................................... 1534
LLayer_GetName ..................................... 1538
LLayer_GetNext ..................................... 1535
LLayer_GetParameters .................................................. 1540
LLayer_GetParametersEx1512 .......................................... 1543
LLayer_GetParametersEx830 ........................................ 1541
LLayer_GetRenderingAttribute ...................................... 1581
LLayer_GetRenderingObjectName .................................. 1583
LLayer_GetRho .......................................................... 1550
LLayer_GetSpecial ...................................................... 1554
LLayer_IsDerived ....................................................... 1561
LLayer_MoveLayer ...................................................... 1556
LLayer_New .............................................................. 1530
LLayer_PrecedingLayer ............................................... 1536
LLayer_PrecedingLayerEx99 ......................................... 1537
LLayer_SetCap ........................................................... 1549
LLayer_SetCurrent ....................................................... 1553
LLayer_SetDerivedParameters ..................................... 1566
LLayer_SetDerivedParametersEx00 ................................. 1567
LLayer_SetName .......................................................... 1539
LLayer_SetParameters .................................................. 1544
LLayer_SetParametersEx1512 ......................................... 1547
LLayer_SetParametersEx830 .......................................... 1545
LLayer_SetRenderingAttribute ................................... 1582
LLayer_SetRenderingAttribute ................................... 1582
LLayer_SetRho ........................................................... 1551
LLayer_SetSpecial ....................................................... 1555
LLayerParam ............................................................. 1756
LLayerParam_Ex1512 ..................................................... 1759
LLayerParam_Ex830 ...................................................... 1757
LLayerViewStatus .......................................................... 1762
LLen ................................................................. 1763
LMacro_BindToHotKey ..................................................... 1101
LMacro_BindToMenu ....................................................... 1102
LMacro_BindToMenuAndHotKey_v9_30 .............................. 1103
LMacro_GetNewTCell ...................................................... 1112
LMacro_IsLoaded .......................................................... 1108
LMacro_Load ............................................................. 1109
LMacro_LoadEx1200 ....................................................... 1110
LMacro_Register .......................................................... 1100
LMacro_Unload ............................................................. 1111
LMagnification ............................................................ 1764
LMarker ................................................................. 1765
LMarkerParam ............................................................ 1766
LMarkerParam_V11 ........................................................ 1767
LObject ................................................................. 1769
LObject_Area ............................................................... 1383
LObject_ChangeLayer ...................................................... 1389
LObject_ConvertToPolygon .......................................... 1391
LObject_Copy ............................................................. 1392
LObject_Delete ............................................................ 1373
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LObject_DistanceToPoint</td>
<td>1395</td>
</tr>
<tr>
<td>LObject_GetGDSIIDataType</td>
<td>1386</td>
</tr>
<tr>
<td>LObject_GetGeometry</td>
<td>1381</td>
</tr>
<tr>
<td>LObject_GetInstance</td>
<td>1378</td>
</tr>
<tr>
<td>LObject_GetLayer</td>
<td>1385</td>
</tr>
<tr>
<td>LObject_GetList</td>
<td>1374</td>
</tr>
<tr>
<td>LObject_GetMbb</td>
<td>1379</td>
</tr>
<tr>
<td>LObject_GetNext</td>
<td>1375</td>
</tr>
<tr>
<td>LObject_GetPort</td>
<td>1482</td>
</tr>
<tr>
<td>LObject_GetShape</td>
<td>1380</td>
</tr>
<tr>
<td>LObject_GetVertexList</td>
<td>1382</td>
</tr>
<tr>
<td>LObject_Perimeter</td>
<td>1384</td>
</tr>
<tr>
<td>LObject_SetGDSIIDataType</td>
<td>1387</td>
</tr>
<tr>
<td>LObject_SnapToGrid</td>
<td>1393</td>
</tr>
<tr>
<td>LObject_SnapToMfgGrid</td>
<td>1394</td>
</tr>
<tr>
<td>LObject_Transform</td>
<td>1376</td>
</tr>
<tr>
<td>LObject_Transform_Ex99</td>
<td>1377</td>
</tr>
<tr>
<td>LOrientation</td>
<td>1770</td>
</tr>
<tr>
<td>LOrientation_Ex99</td>
<td>1771</td>
</tr>
<tr>
<td>LOutlineStyle</td>
<td>1772</td>
</tr>
<tr>
<td>LOverwriteCellsScopeOnImport</td>
<td>1705</td>
</tr>
<tr>
<td>LPalette</td>
<td>1774</td>
</tr>
<tr>
<td>LPass</td>
<td>1775</td>
</tr>
<tr>
<td>LPass_GetList</td>
<td>1577</td>
</tr>
<tr>
<td>LPass_GetNext</td>
<td>1578</td>
</tr>
<tr>
<td>LPass_GetParameters</td>
<td>1579</td>
</tr>
<tr>
<td>LPass_New</td>
<td>1576</td>
</tr>
<tr>
<td>LPass_SetParameters</td>
<td>1580</td>
</tr>
<tr>
<td>LPassMode</td>
<td>1776</td>
</tr>
<tr>
<td>LPassParam</td>
<td>1777</td>
</tr>
<tr>
<td>LPassType</td>
<td>1778</td>
</tr>
<tr>
<td>LPie_CreateNew</td>
<td>1422</td>
</tr>
<tr>
<td>LPie_GetParams</td>
<td>1424</td>
</tr>
<tr>
<td>LPie_SetParams</td>
<td>1425</td>
</tr>
<tr>
<td>LPieParams</td>
<td>1779</td>
</tr>
<tr>
<td>LPoint</td>
<td>1780</td>
</tr>
<tr>
<td>LPoint_Add</td>
<td>1665</td>
</tr>
<tr>
<td>LPoint_Set</td>
<td>1664</td>
</tr>
<tr>
<td>LPoint_Subtract</td>
<td>1666</td>
</tr>
<tr>
<td>LPoint_Transform</td>
<td>1667</td>
</tr>
<tr>
<td>LPoint_Transform_Ex99</td>
<td>1668</td>
</tr>
<tr>
<td>LPolygon_CircleToPolygon</td>
<td>1448</td>
</tr>
<tr>
<td>LPolygon_HasCurve</td>
<td>1449</td>
</tr>
<tr>
<td>LPolygon_New</td>
<td>1446</td>
</tr>
<tr>
<td>LPolygon_RemoveAllCurves</td>
<td>1450</td>
</tr>
<tr>
<td>LPolygon_StraightenAllCurves</td>
<td>1451</td>
</tr>
<tr>
<td>LPolygon_WireToPolygon</td>
<td>1447</td>
</tr>
<tr>
<td>LPort</td>
<td>1781</td>
</tr>
</tbody>
</table>
Chapter 36: Alphabetical List of UPI and LComp Functions

LPort_AddObject .................................................. 1480
LPort_Delete ....................................................... 1454
LPort_DeletePortOnly ............................................. 1484
LPort_Find .......................................................... 1455
LPort_FindNext ....................................................... 1456
LPort_GetAccessDirection ....................................... 1488
LPort_GetDefaultTextAlignment ................................. 1472
LPort_GetLayer ...................................................... 1462
LPort_GetList ......................................................... 1457
LPort_GetMbb ........................................................ 1463
LPort_GetNext ......................................................... 1458
LPort_GetObjectsList .............................................. 1478
LPort_GetObjectsNext .............................................. 1479
LPort_GetRect ........................................................ 1464
LPort_GetTerminalType ........................................... 1486
LPort_GetText ........................................................ 1459
LPort_GetTextAlignment .......................................... 1468
LPort_GetTextSize .................................................. 1461
LPort_IsLegacy ....................................................... 1483
LPort_New ............................................................. 1453
LPort_New_With_Object ............................................ 1477
LPort_RemoveObject ............................................... 1481
LPort_RemoveObject ............................................... 1482
LPort_RemoveObject ............................................... 1488
LPort_Set ............................................................. 1465
LPort_SetAccessDirection ....................................... 1487
LPort_SetDefaultTextAlignment ................................. 1473
LPort_SetTerminalType ........................................... 1485
LPort_SetText ........................................................ 1460
LPort_SetTextAlignment .......................................... 1470
LPort_SetTextSize .................................................. 1466
LPropertyType ......................................................... 1782
LRect ................................................................. 1783
LRect_Set ............................................................ 1670
LRect_Transform .................................................... 1671
LRect_Transform_Ex99 .............................................. 1672
LRenderingAttribute .............................................. 1784
LRenderingAttributeIndex ....................................... 1786
LRenderingMode ..................................................... 1787
LSelection .......................................................... 1788
LSelection_AddAllObjectsInRect ............................... 1508
LSelection_AddAllObjectsOnLayer .............................. 1506
LSelection_AddObject .............................................. 1503
LSelection_ChangeLayer ......................................... 1513
LSelection_Clear ..................................................... 1500
LSelection_Copy ..................................................... 1496
LSelection_Copy ..................................................... 1497
LSelection_Copy ..................................................... 1498
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSelection_Cut</td>
<td>1495</td>
</tr>
<tr>
<td>LSelection_DeselectAll</td>
<td>1502</td>
</tr>
<tr>
<td>LSelection_Duplicate</td>
<td>1515</td>
</tr>
<tr>
<td>LSelection_Flatten</td>
<td>1518</td>
</tr>
<tr>
<td>LSelection_FlipHorizontal</td>
<td>1520</td>
</tr>
<tr>
<td>LSelection_FlipVertical</td>
<td>1521</td>
</tr>
<tr>
<td>LSelection_GetLayer</td>
<td>1512</td>
</tr>
<tr>
<td>LSelection_GetList</td>
<td>1510</td>
</tr>
<tr>
<td>LSelection_GetNext</td>
<td>1511</td>
</tr>
<tr>
<td>LSelection_GetObject</td>
<td>1505</td>
</tr>
<tr>
<td>LSelection_Group</td>
<td>1516</td>
</tr>
<tr>
<td>LSelection_Merge</td>
<td>1519</td>
</tr>
<tr>
<td>LSelection_Move</td>
<td>1514</td>
</tr>
<tr>
<td>LSelection_PasteToLayer</td>
<td>1499</td>
</tr>
<tr>
<td>LSelection_RemoveAllObjectsInRect</td>
<td>1509</td>
</tr>
<tr>
<td>LSelection_RemoveAllObjectsOnLayer</td>
<td>1507</td>
</tr>
<tr>
<td>LSelection_RemoveObject</td>
<td>1504</td>
</tr>
<tr>
<td>LSelection_Rotate</td>
<td>1524</td>
</tr>
<tr>
<td>LSelection_RotateAroundPoint</td>
<td>1525</td>
</tr>
<tr>
<td>LSelection_SelectAll</td>
<td>1501</td>
</tr>
<tr>
<td>LSelection_SliceHorizontal</td>
<td>1522</td>
</tr>
<tr>
<td>LSelection_SliceVertical</td>
<td>1523</td>
</tr>
<tr>
<td>LSelection_SnapToMfgGrid</td>
<td>1527</td>
</tr>
<tr>
<td>LSelection_UnGroup</td>
<td>1517</td>
</tr>
<tr>
<td>LSelectionParam</td>
<td>1789</td>
</tr>
<tr>
<td>LShapeType</td>
<td>1790</td>
</tr>
<tr>
<td>LSpecialLayer</td>
<td>1791</td>
</tr>
<tr>
<td>LStatus</td>
<td>1792</td>
</tr>
<tr>
<td>LStatusBar_SetMsg</td>
<td>1093</td>
</tr>
<tr>
<td>LStatusBar_SetMsg</td>
<td>1115</td>
</tr>
<tr>
<td>LStipple</td>
<td>1793</td>
</tr>
<tr>
<td>LTCellParameterType</td>
<td>1796</td>
</tr>
<tr>
<td>LTechnology</td>
<td>1797</td>
</tr>
<tr>
<td>LTechnologyEx840</td>
<td>1798</td>
</tr>
<tr>
<td>LTerminalType</td>
<td>1799</td>
</tr>
<tr>
<td>LTorus_CreateNew</td>
<td>1427</td>
</tr>
<tr>
<td>LTorus_GetParams</td>
<td>1429</td>
</tr>
<tr>
<td>LTorus_SetParams</td>
<td>1430</td>
</tr>
<tr>
<td>LTorusParams</td>
<td>1800</td>
</tr>
<tr>
<td>LTransform</td>
<td>1801</td>
</tr>
<tr>
<td>LTransform_Add</td>
<td>1678</td>
</tr>
<tr>
<td>LTransform_Add_Ex99</td>
<td>1679</td>
</tr>
<tr>
<td>LTransform_Ex99</td>
<td>1802</td>
</tr>
<tr>
<td>LTransform_GetInverse</td>
<td>1682</td>
</tr>
<tr>
<td>LTransform_Set</td>
<td>1674</td>
</tr>
<tr>
<td>LTransform_Set_Ex99</td>
<td>1675</td>
</tr>
<tr>
<td>LTransform_Subtract</td>
<td>1680</td>
</tr>
<tr>
<td>LTransform_Subtract_Ex99</td>
<td>1681</td>
</tr>
</tbody>
</table>
LTransform_Zero ........................................ 1676
LTransform_Zero_Ex99 .............................. 1677
LUpi_GetReturnCode ................................. 1122
LUpi_GetSerialNumber .............................. 1114
LUpi_GetUpdateDisplayMode ...................... 1125
LUpi_InQuietMode ................................... 1117
LUpi_InsertMenuItemSeparator .................... 1120
LUpi_SetDrawingTool ................................. 1119
LUpi_SetQuietMode ................................... 1116
LUpi_SetReturnCode ................................. 1121
LUpi_SetSelectionTool .............................. 1118
LUpi_SetUpdateDisplayMode ...................... 1124
LVertex ........................................... 1803
LVertex_Add ......................................... 1402
LVertex_AddCurve .................................. 1404
LVertex_Delete ...................................... 1403
LVertex_GetArray ................................... 1398
LVertex_GetCount ................................... 1397
LVertex_GetCurve ................................... 1405
LVertex_GetCurveEX ................................. 1406
LVertex_GetCurveExactCenter ..................... 1407
LVertex_GetNext ..................................... 1399
LVertex_GetPoint .................................... 1400
LVertex_HasCurve ................................... 1408
LVertex_RemoveCurve ................................. 1410
LVertex_SetCurve .................................... 1409
LVertex_SetPoint .................................... 1401
LWindow ........................................... 1804
LWindow_Close ....................................... 1134
LWindow_CloseAll .................................... 1135
LWindow_EditInPlacePopOut ....................... 1137
LWindow_EditInPlacePopToTop ..................... 1138
LWindow_EditInPlacePushIn ....................... 1136
LWindow_GetCell ..................................... 1141
LWindow_GetEditTransform ......................... 1142
LWindow_GetFile ..................................... 1140
LWindow_GetList ..................................... 1130
LWindow_GetName ..................................... 1151
LWindow_GetNext ..................................... 1131
LWindow_GetParameters ............................. 1144
LWindow_GetText ..................................... 1149
LWindow_GetTopCell ................................ 1143
LWindow_GetType ..................................... 1139
LWindow_GetVisible ................................ 1129
LWindow_GetWindowHandle ......................... 1145
LWindow_IsLast ....................................... 1132
LWindow_LoadTextFile ............................... 1147
LWindow_MakeVisible ............................... 1133
LWindow_NewTextWindow ........................................ 1146
LWindow_SaveToFile ............................................. 1148
LWindow_SetName .................................................. 1152
LWindow_SetText .................................................. 1150
LWindowType ....................................................... 1805
LWire_GetCapType ................................................ 1435
LWire_GetJoinType ............................................... 1436
LWire.GetLength ................................................. 1438
LWire_GetMiterAngle ........................................... 1437
LWire_GetResistance .......................................... 1440
LWire_GetSquares ............................................... 1439
LWire_GetWidth ................................................ 1434
LWire_New ....................................................... 1433
LWire_SetCapType ............................................... 1443
LWire_SetJoinType ............................................. 1442
LWire_SetMiterAngle ........................................ 1444
LWire_SetWidth ................................................ 1441
LWireConfig ....................................................... 1806
LWireConfigBits ................................................ 1807
LWireParam ....................................................... 1808
Port Access Direction .......................................... 1799
technology ...................................................... 1795
Upi_LPGetGetMustConnect ...................................... 1489
Upi_LPGetGetMustConnect ...................................... 1490
Upi_LPGetGetPinName .......................................... 1491
UPIDrawingToolType ........................................... 1809
Appendix

Legacy Tools
Introduction to Placement and Routing

Placement and Routing in L-Edit

This volume of the L-Edit user guide describes the automatic standard cell place and route (SPR) features of L-Edit. The chapter “Placing and Routing Standard Cells” on page 1915 explains the three SPR modules: core generation, pad routing, and padframe generation. It also explains global signal routing, which is used to route as many as two I/O signals independent of other signals. The chapter “Standard Cell Library” on page 1960 provides design rules for creating standard cell libraries.

Syntax and usage for the following file formats are detailed in the chapter “Place and Route File Formats” on page 1969:

- CAP files—an SPR output file that lists the capacitance, area, and length of each node due to routing.
- EDIF files—a netlist format used as input for SPR.
- TPR files—a netlist format used as input for SPR.
- SDF file—an output file that contains interconnect delays due to routing in standard delay format.

Standard Cell Place and Route (SPR)

Standard cell place and route (SPR) is a place and route package for standard cells that can automatically lay out entire chips. It consists of three modules: a core place and route module to generate a core cell, a padframe generator, and a pad route module to connect the padframe with the core cell. You can run these three modules individually or together.

SPR uses standard cells and pad cells from a standard cell library. Netlists can be provided in Electronic Design Interchange Format (EDIF) or Tanner Place and Route (TPR) format. If needed, a mapping tool allows you to achieve consistency between the cell and pin names in your netlist and your library.

SPR generates a core, a padframe, and a chip cell in L-Edit which then can be checked for design rules and extracted. In order to verify the delay constraints, you can generate a nodal capacitance (CAP) file or a standard delay format (SDF) file, or both, during the place and route step.

The place and route steps are fully automated. You can use two- or three-layer routing, with the latter including the option of over-the-cell (OTC) routing. Up to two I/O signals (e.g., clock signals) can be routed separately to better control delay and skew when you use the global input signal routing function. Among the many features of SPR are standard cell grouping (cell clustering) and critical nets consideration.
A2  Placing and Routing Standard Cells

Introduction

L-Edit SPR places and routes a design using a user-provided EDIF or TPR netlist and a standard cell source library as input. SPR includes three options:

- **Core place and route** generates a core cell using standard cells from a standard cell library. Parameters for standard cell core place and route can be specified in the dialog “**SPR Core Setup**” (page 1926).

- **Padframe generation** creates a user-specified padframe with pad cells from the standard cell library. Parameters for padframe generation are specified in the dialog “**SPR Padframe Setup**” (page 1937).

- **Pad routing** routes signals, including power and ground, between the layout core of a chip and its padframe. Parameters for pad routing are specified in the dialog “**SPR Pad Route Setup**” (page 1941).

You can perform the three SPR operations in one step or separately. L-Edit can perform a pad route against a single pregenerated core cell or a set of core cells and/or other customized building blocks, as long as the layout core of the chip is composed in one cell. Similarly, L-Edit can perform a pad route against a pregenerated padframe, which can also be built in any manner, as long as it forms a single padframe cell and conforms to SPR constraints.
Running all three operations produces a completed design like the following:

![Completed design diagram]

**Required Files**

To run SPR, the following files are required:

- A design file (.tdb).
- A netlist file (.tpr, .edf, .edn, or .edi). This file contains a textual description of your schematic design and identifies the cells that are required from the standard cell library file.
- A standard cell library such as `morbn20d.tdb`, which contains the standard cells and pad cells required for your design. This particular file is a component of the Tanner Research standard cell library `SCMOSLib`. You can also create your own standard cell library.

**Note:** To place and route a design, you must first define (in your design file) a technology setup appropriate to your standard cell library. If you start L-Edit with an empty design file, use **File > New** to copy the technology setup from an existing TDB file or a technology file such as `morbn20.tdb` before setting up or running SPR. Alternatively, you can simply open a design file that already contains the correct technology setup.

**SPR Process Overview**

To place and route a design using L-Edit/SPR, you will typically perform the following steps:

- Create a schematic representation of your design.
Export the schematic as either a flattened EDIF netlist or in TPR format. L-Edit supports EDIF version 2.0.0, EDIF level 0, keywordLevel 0, viewType NETLIST, or netlists with one level of hierarchy.

Launch L-Edit. Use File > New to create your design file (layout file). Import the design information (technology setup) from your cell library into the design file by entering your cell library file name in the Copy TDB setup from file field of the New File dialog.

Use File > Save to label and save your initial design file with an appropriate name.

Choose Tools > SPR > Setup. In the SPR Setup dialog (see “SPR Setup” on page 1923) specify the names of the standard cell library file and the netlist file. Also specify the power and ground node and port names as used in your schematic. (These names must match the names of the power and ground ports in the standard cells.)

Click the Initialize Setup button. This will read the netlist and initialize the following setup dialogs with netlist information: critical nets, I/O signals, padframe layout, and core and padframe signals of the pad route.

Click the buttons Core Setup, Padframe Setup, and Pad Route Setup, respectively, to specify the remaining setup parameters for core placement and routing (see “SPR Core Setup” on page 1926), padframe generation (see “SPR Padframe Setup” on page 1937), and pad routing (see “SPR Pad Route Setup” on page 1941).

Choose Tools > SPR > Place and Route. Select the appropriate option (Core place and route, Padframe generation, or Pad route) singly or in any combination. Depending on your standard cell design, uncheck or check the Global input signal routing option. (Global input signal routing requires special buses to be available in your standard cells, see “Global Input Signal Routing (Clock Routing)” on page 1921 for further details). Decide on your core configuration. For example, select Square if you want to obtain a square core shape. Check or uncheck the placement and routing optimization options. Specify the output options—for example, whether you want to label nodes with ports (to support node recognition during extraction) or whether you want to generate files containing nodal capacitances.

Click the Run button. Depending on your selected options, SPR will generate up to three new cells: a core cell, a padframe cell, and/or a chip cell (which contains the core, the padframe and the padroute). If these cells already exist in your design file, SPR will prompt you before overwriting them.

When processing is complete, SPR will output an SPR Complete dialog providing summary statistical information for your design. (You can use Tools > SPR > Summary at any time to display a text file with further details.)

Click the OK button in the summary dialog to display the completed design. The example shown below includes 990 standard cells (3,510 gates). On a 450 MHz Pentium II PC with 128 MB RAM, using both placement and routing optimization, SPR can generate this design in less than ten minutes.
Confirm that the dimensions of the core and/or padframe fall within the size limitation imposed by your vendor. If not, you need to re-run SPR with either a different core configuration or increased placement and routing optimization (see “Placement Optimization” on page 1951).

Verify the design using L-Edit DRC (see “Running DRC” on page 550) and L-Edit Extract (see “Extracting Layout” on page 912).

Save the design in GDSII format and send it to your vendor for fabrication.

**Design Tips**

If you use an EDIF netlist and your netlist cell and/or port names differ from the names used in the standard cell library, use the Mapping Table button on the SPR Setup dialog (see “Mapping Table” on page 1924) to generate a mapping table that allows you to assign the correlating names. SPR will use this mapping information when it discovers a discrepancy between a cell or port name in your netlist and your cell library.

Before running SPR on a new layout, use the Initialize Setup button (see “ Initializing Setup” on page 1926) to automatically enter pad-related information (for example, I/O signal configuration) from your netlist into the setup dialogs. SPR will only refer to the netlist if a dialog is empty. In this case, L-Edit will automatically fill the dialog fields with the netlist information.

We recommend that new users first generate a core separately. The dialogs that require I/O signal information (SPR Core Setup—I/O Signals) can be filled out either manually or, if pads or I/O signals are available in the netlist, initialized with the netlist information using the Initialize Setup feature. The padframe should be created next, taking the core dimensions into consideration. Finally, you can generate the new chip cell by performing a pad route using the core and padframe cells.
Core Generation and Pad Routing

In pad routing, L-Edit routes signals only between the outer edge of the core and the inner edge of the padframe. Therefore, the core’s position and dimensions are critical, but its internal geometry is not.

L-Edit determines the core’s position by searching for an abutment port, which defines the edges of the core cell. The program creates the abutment port on the layer defined as the Icon layer. To define the Icon layer, choose Setup > Special Layers.

The core and pad routing must adhere to the following constraints:

 The core must contain signal ports along its edges for every signal going to the padframe.
 Signal ports on the core and padframe must be ordered such that no signal running between core and padframe crosses over another signal, except for power and ground.
 However, signals may cross power or ground rails only if the materials used for routing are different (for example, Metal2 for I/O signals and Metal1 for power and ground).
 Power and ground lines are of the same material and may not cross.

The following figure illustrates the placement of signal ports along the core.

The pad router can only route a single core to the padframe. To use several core cells, you must create a new cell, instance each core cell, manually wire the instances together and finally surround the contents of this new cell with an abutment port and signal ports as described above. L-Edit then treats this new cell as a “single” core for the purposes of pad routing. You specify the name of this newly created core cell and other necessary information in the dialogs “SPR Core Setup–General” (page 1927) and “SPR Pad Route Setup–General” (page 1943).

A netlist is not required if you perform a separate pad route with pregenerated core and padframe cells.

Padframe Generation and Pad Routing

In padframe routing, L-Edit routes signals only as far as the inner edge of the padframe. Therefore, the padframe’s position and dimensions are critical, but its internal geometry is not. To indicate the region
in which the core may be placed, L-Edit places a rectangular padframe port on the inner edge of the padframe.

For each signal going to the core, the padframe must contain one signal port along its inner edge. Signal ports for each signal going to the core must be placed on the padframe in the same order and on the same side as the signal ports around the core cell. These ports may be at the top level (in the padframe cell itself) or they may be one level lower in the hierarchy (in a pad cell instanced by the padframe). Power and ground pads must be on different sides of the padframe. L-Edit cannot route directly between the pads on the padframe—it can only route between the padframe and the core.

The following illustration shows a padframe with ports for signals, power, and ground.

![Padframe with ports for signals, power, and ground. The padframe port defines the inner edge of the padframe.](image)

**SPR Port Annotation**

The Padframe generator of SPR will create a new port for each pad, and give these ports the same name as the netlist instance name of the corresponding pad. These new ports are particularly useful for extract/simulation, so that the I/O pads of the resulting netlist have constant, persistent names. They are also useful in creating bonding diagrams, and other documentation.

These ports are created in the padframe cell. These new ports are placed coincident with the ports named “Pad” inside each pad library cell, on the same layer as the “Pad” port, with text size that is 10 times as large. If the pad cell does not contain a “Pad” port, no new port is created.

A padframe can be generated using two methods:
Generating a Padframe from a Netlist with Pad Cells

☑ If the netlist contains pad cells, use the Initialize Setup button to automatically include pad cell instance names and their location in the SPR Padframe Setup—Layout dialog.

☑ Fill out the remaining input fields, like padframe size and padframe cell name. In the SPR Setup dialog, provide the names of the cell library that contains the pad cells and the appropriate netlist.

☑ Run SPR with Padframe generation turned on.

Generating a Padframe Without a Netlist or Without Pad Cells

☑ If the netlist does not contain pad cells or if no netlist is available, manually input pad cell names and their location in the SPR Padframe Setup—Layout dialog.

☑ Fill out the remaining input fields, like padframe size and padframe cell name.

☑ In the SPR Setup dialog, provide the name of the cell library that contains the pad cells and leave the netlist input field blank.

☑ Run SPR with Padframe generation turned on.

Global Input Signal Routing (Clock Routing)

Global input signal routing is used to route as many as two I/O signals, such as clock nets, independently from other signals.

For global input signal routing, standard cells must contain two global signal buses, with four global signal ports, placed above and below the power and ground buses. The following illustration shows a standard cell (top view) with two global signal buses. GS1 and GS2 are global signal ports of this standard cell.

![Standard cell (top view) with two global signal buses. GS1 and GS2 are global signal ports of this standard cell.](image)
During core routing, L-Edit connects signal ports belonging to global signal nets (labeled A and B in the illustration below) to the dedicated global signal bus.

After placement, L-Edit adds buffer cells to both ends of the standard cell rows. L-Edit calculates the number of buffer cells required for each row by dividing the number of standard cells connected to the global signal nets by the driving force, which the user specifies in the dialog “SPR Core Setup–Global Signals” (page 1934).

Buffer cells are dedicated to one of the two global signal buses. Each buffer cell contains an IN port that is accessible from the side and placed on the vertical layer. The IN port of the outermost buffer cell is connected with the vertical global signal rail on this side.

The vertical global signal rail is placed on the vertical layer, inside of the vertical power rail. It is twice as wide as the IN port of the buffer cell(s) on this side.

Pad routing connects the vertical global signal rails to the appropriate ports on the padframe. Layer assignment for global signal pad routing is equivalent to the layer assignment for regular I/O signals leaving the core. The pads of the global signal nets have to be located either on the left (for the left global signal rail) or on the right side (for the right global signal rail).
Before running SPR, you must set the appropriate options. Use Tools > SPR > Setup to open the SPR Setup dialog.

Specify the following:

**Standard cell library file**

File containing the standard cells and pad cells that L-Edit uses to generate your design. If your design file already contains the required cells, you may use it instead of the standard cell library. Enter the full pathname if the file is not in the current L-Edit directory.
Netlist file

File containing a textual description of your schematic design, which identifies the cells required from the standard cell library. This file is always required when you place and route a core. It is optional when you perform only pad routing or padframe generation.

SPR accepts flattened EDIF netlists or netlists with one level of hierarchy only.

Two types of netlist files are supported:

- TPR—Tanner Place and Route Format, produced by S-Edit
- EDIF—EDIF version 2.00, EDIF level 0, keywordLevel 0, viewType NETLIST. (Acceptable filename extensions are .edf, .edn, and .edi.)

A SPICE netlist can also be used if it is first converted to TPR format.

Mapping Table

Accesses the dialog “Mapping Table” (page 1924) to map cell and port names between the EDIF netlist and the standard cell library.

Initialize Setup

Reads pad-related information from the netlist and completes the fields in the setup dialogs that specify critical nets, padframe layout, core signals, and padframe signals. If these fields already contain information, SPR will prompt you to keep or overwrite the values. (See “Initializing Setup” on page 1926.)

Remove power and ground nets from the netlist and power and ground pins from the gates during reading

If power and ground pins are explicitly placed on schematic symbols, this option should be checked to remove power and ground from the netlist for correct place and route performance.

Power signal

Schematic netlist name of the power node. The power signal must have the same name as the power port in the standard cells.

Ground signal

Schematic netlist name of the ground node. The ground signal must have the same name as the ground port in the standard cells.

When you have typed the correct information in these fields, click the appropriate button—Core setup, which opens “SPR Core Setup–General” (page 1927), Padframe setup, which opens “SPR Padframe Setup–General” (page 1937), or Pad route setup, which opens “SPR Pad Route Setup–General” (page 1943)—to continue SPR setup.

Each of these dialogs contains a Reset button, which resets all fields and options to the values they held when you accessed the dialog.

If you are using an EDIF netlist and it has different cell and port names than those used in the cell library, you must map these names correctly. For additional information on this topic, see the section “Mapping Table,” below.

Mapping Table

L-Edit invokes the Mapping Table dialog while processing the EDIF netlist whenever it finds a discrepancy between the cell or port names. You can also click Mapping Table in the SPR Setup dialog to directly generate a mapping table before running SPR.
Mapping information is saved in the design file. If you change the netlist (within the same design file), the mapping table will display values for previously mapped cells and ports.

Use this dialog to define the correspondence between:

- Cells in the EDIF netlist and cells in the standard cell library file.
- Ports in the EDIF netlist and ports in the standard cell library file, within individual cells.

Port mapping is only required if a port name discrepancy occurs. However, if you map one port in an individual cell, you must map all ports in that cell.

If your cell interface in the EDIF file contains ports which are not connected in your design, you can label them as “not used” during the mapping process.

To map a cell, click the netlist or TDB cell name, or **Browse Cells**. To map a port, click the netlist or TDB port name, or **Browse Ports**. L-Edit displays a dialog in which you select the correct cell or port.

**Cell mapping**
A numbered list of cells named in the EDIF netlist.

- **Netlist cell names**—list of cells named in the EDIF netlist.
- **TDB cell names**—list of cells contained in the standard cell library file.

To map a cell, click on a cell name, or **Browse Cells**. L-Edit displays a dialog where you can select the correct cell.

**Port mapping**
A numbered list of ports named in the specified cell in the EDIF netlist.
Chapter A2: Placing and Routing Standard Cells

Initializing Setup

The **Initialize Setup** function keeps the setup dialogs and the netlist synchronized. You should use it when you create a new design and whenever the netlist is changed or updated. When you click the **Initialize Setup** button, SPR updates the following dialog values with values from the netlist:

- Critical nets in **SPR Core Setup—Placement**, if any
- I/O signal specifications in **SPR Core Setup—I/O Signals**
- Pad route specifications in **SPR Pad Route Setup—Padframe Signals** and **SPR Pad Route Setup—Core Signals**
- Padframe specifications in **SPR Padframe Setup—Layout**

You cannot cancel or undo this operation.

**SPR Core Setup**

In this dialog, you define the parameters L-Edit will use to generate your design core. The dialog consists of seven tabs:

- **General**
- **Layers**
- **Design Rules**
- **Placement**
- **Global Signals**
- **Power**
- **I/O Signals**

Each tab contains a **Reset** button, which resets all fields and options to the values they held when you accessed the dialog.
SPR Core Setup—General

The **General** tab contains fields used to define the library cells, standard cell ports, and output cells used in generating the core.

L-Edit requires three special standard cells in a library set: a *row crosser cell*, the *tie-to-power* cell, and the *tie-to-ground* cell. They are used for node connections only and are not included in the netlist.

Specify the following:

**Library cell names**

- **Row crosser cell**: This cell contains one row cross port and is placed to make up a cross-row pass to route wires across a standard cell.
- **Tie-to-Power** and **Tie-to-Ground** cells: These cells are needed where a standard cell has a pin directly tied to Vdd or Gnd, respectively.

For detailed design information on these cells, see “Special Standard Cells” on page 1963.

**Port names in standard cells**

- **Row crosser port**
- **Abutment port**

**Core cell**

The name of the core cell to be created.

**Row base name**

The base name of the row cells to be created.


**SPR Core Setup—Layers**

The Layers tab contains fields that define the layers L-Edit will use to route the core. You use it to specify whether two or three layers are used for routing. If you use three layers, you can also select over-the-cell (OTC) routing.

This tab also contains fields for the layer-to-layer capacitance between routing layers. These capacitance values are used for extracting nodal capacitances, which are written to the CAP file (see “Nodal Capacitance Files (CAP)” on page 1953).

Click on **2-layer routing** or **3-layer routing** to choose a routing configuration. If you use three-layer routing, **Over-the-cell routing** will be an available option.

Specify the following:

**Signal routing layers**

Specify the routing layers and the via layer(s) to be used for channel routing.

**Power/Ground routing layers**

Specify the routing layers for power and ground. This assignment must be consistent with the layer assignment of the power and ground buses within the standard cells (usually in the H1 layer).

**Layer-to-layer area capacitances**

Enter the layer-to-layer capacitances between your routing layers (in aF/sq. micron). These values are only required if the **Write CAP file** option in the Standard Cell Place and Route dialog is checked.
Over-the-Cell Routing

Over-the-cell (OTC) routing uses tracks above the cells, in the H2 layer, between the channel edge and the “outermost” port inside the cells, for routing.

A special algorithm sorts net segments to utilize these tracks as effectively as possible. The number of OTC tracks depends directly on how you have placed your ports inside the standard cells. If all ports are lined up in the center of the standard cell, a maximum number of OTC tracks can be utilized.

Layout example of three metal layers with OTC routing.
SPR Core Setup – Design Rules

Use the **Design Rules** tab to specify the design rules L-Edit must follow to route the core in conformance with the technology used to fabricate your design.

The figures “SPR widths, spaces, and via surrounds in 2-layer routing” on page 1931 and “Minimum cell-cell and cell-power spaces” on page 1931 illustrate the application of design rules in 2-layer routing. The letters in the dialog fields provide a key for the labels in the illustrations and the values they represent.

This tab also displays the effective channel pitch on the H layer(s) in a read-only field. (The channel pitch is the distance between the centerlines of two neighboring horizontal routing segments.) This value is internally calculated according to your design rules.
SPR widths, spaces, and via surrounds in 2-layer routing

Minimum cell-cell and cell-power spaces
SPR Core Setup–Placement

This tab contains options for controlling the outcome of the automatic placer.

Specify the following options:

**Net Name**
- Enter a net name.

**Criticality**
- Enter an integer criticality value. A positive integer value denotes a higher priority net, a negative integer value reduces the importance of the net during placement. See “Assigning Net Criticality” on page 1932 for further details.

To add a new critical net, click **Add**. To delete a critical net, select it and click **Delete**.

**Instance Name**
- Instance name of a standard cell to be included in a cluster.

**Cluster**
- Enter an integer value. All cell instances with the same cluster number are placed within one cluster. See “Clustering Standard Cells” on page 1933 for further details.

To add a new cell instance to a cluster, click **Add**. To delete a cell instance from a cluster, highlight the instance name and click **Delete**.

**Assigning Net Criticality**

Nets that are critical in your layout can be specified in the **List of critical nets**. Criticality is expressed as an integer value that may be positive or negative. The higher the criticality value, the higher the priority of the net during placement. (A positive value net is given a higher priority, a negative value net is given a lower priority during placement.) The value for any net that is not specified in this table is zero.
The consideration of net criticality in L-Edit/SPR is based on two assumptions:

- The numerical value of the criticality describes the relative importance of a net compared to others. For example, if the criticality of net A is twice the criticality assigned to net B, then the placer considers it as twice as important to reduce the length of net A compared to net B.
- The critical values are scaled internally according to the largest value that has been entered, with the largest value assigned to a fixed internal value. Hence, if net A is the only net with an assigned criticality, than any criticality value greater than zero for this net would lead to the same result.

Net criticality can be entered either by using the EDIF netlist attribute `criticality`, or directly in the SPR Core Setup—Placement dialog. If SPR finds criticality values in this dialog, it will ignore any criticality values found in the netlist. To transfer criticality values from the netlist into this dialog, use the Initialize Setup button (see “Initializing Setup” on page 1926).

**Clustering Standard Cells**

The List of clusters allows you to group standard cells together. All cell instances assigned the same cluster number are placed side by side, from left to right, in the order in which they appear in this table. Cell clusters cannot be “broken”—they must fit on one row. Note that row crossers might be inserted between two cells within one cluster.

**Note:**

If you turn off placement optimization, cell clustering can also be used to place your cells in a specific sequence. SPR always places cell instances from left to right inside a cluster according to their sequence in the list of clusters. Without placement optimization, clusters are placed according to their number, starting with the lowest row, from left to right (see the figure on the following page). Cell instances that are not included in any cluster are placed subsequently, in the sequence of the netlist.
### SPR Core Setup—Global Signals

This tab contains options used to route global input signals. If you do not check the option **Global input signal routing** in the **Standard Cell Place and Route** dialog (see “Standard Cell Place and Route” on page 1948), you can ignore this dialog.

![SPR Core Setup dialog](image)

Specify the following options:

<table>
<thead>
<tr>
<th><strong>Global input signal names in netlist</strong></th>
<th>Specify one or two signals for global routing.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ports names in standard cell</strong></td>
<td>Names of the bus ports in the standard cells that will be used for each global input signal. This port name assignment subsequently defines the bus, rail, and pad positions used to route each global signal net (see “Global Input Signal Routing (Clock Routing)” on page 1921).</td>
</tr>
<tr>
<td><strong>Buffer cells in standard cell library</strong></td>
<td>Names of the buffer cells to be placed on the left and right side of the standard cell rows. If your design has only one global input signal, specify one buffer cell on the same side as the global input signal rail.</td>
</tr>
<tr>
<td><strong>Driving force</strong></td>
<td>Driving force is the driving capability (fanout) of one buffer cell—the maximum number of standard cells that can be driven by this buffer cell. The value must be greater than or equal to 1. L-Edit calculates the number of buffer cells to place on the edge of each standard cell row by dividing the number of driven cells in the row by this value.</td>
</tr>
<tr>
<td><strong>Global input signal 1 rail on</strong></td>
<td>Click the option button for whether the <strong>Global input signal 1 rail</strong> should be on the <strong>Left side</strong> or the <strong>Right side</strong> of the core. L-Edit will place the rail for <strong>Global input signal 2</strong> on the opposite side.</td>
</tr>
</tbody>
</table>
Note: The assignment of global signal bus ports to each global signal net determines the assignment of these nets to either the upper or lower global signal bus. Because buffer cells are specifically connected to either the upper or lower global signal bus, this port assignment also determines which global signal net the left and right global signal rails represent.

SPR Core Setup–Power

This tab contains options for the placement and width of power and ground rails.

Specify the following options:

**Power (Vdd) rail on:** Power and ground rails can be placed either on the left or right side of the core cell. This choice will be overwritten by the location of the power and ground pads if the padframe is generated simultaneously and a conflict is detected.

**Automatic (Bus width x fanout):** Check here if you want your power and ground rail widths to be calculated internally by SPR. In this case, the width is determined by multiplying the bus width (in the rows) with the number of rows to be driven.

**Power (Vdd) rail width:** The width of the Vdd rail in display units (if Automatic is unchecked).

**Ground (Gdd) rail width:** The width of the Gnd rail in display units (if Automatic is unchecked).
**SPR Core Setup–I/O Signals**

This tab contains options for the location of input/output signals around the core. **Initialize Setup** will complete this dialog automatically if your netlist contains pad connections or interface I/O signals (for EDIF netlists only; see “EDIF Files” on page 1970).

You do not fill out the **I/O signal list** if you perform core place and route in conjunction with pad routing and padframe generation.

Specify the following options:

**I/O signals leave core on**

Layer (H1 or V) on which I/O signals will leave the core. Options are **H1 layer** and **V layer**. If you perform pad routing, this layer must be identical to the pad routing layer for I/O signals (see “SPR Pad Route Setup–Layers” on page 1943).

**I/O signal list**

Defines the name of each I/O signal, the side from which it exits, and its relative position on that side.

Use the **Position** value to specify the relative position of a signal on a given side proceeding counter-clockwise. The higher the value, the later a signal’s position on a side. For each side, signals are ordered as follows:

- **Left**: top to bottom
- **Bottom**: left to right
- **Right**: bottom to top
- **Top**: right to left

Only the left and right core edges can be used for routing global input signals. It is also recommended that you do not use the uppermost and lowermost pads for global signals.
To add a signal to the list, click **Add**. A **New Signal** is highlighted and can be edited. The name of the signal must be the same as in the netlist. To delete a signal, highlight the signal in the list and click **Delete**.

**SPR Padframe Setup**

In padframe generation, L-Edit takes pad cells from the standard cell library, places them in a rectangular ring, and if required, connects them together. The exact size and shape of this padframe is determined by the *maximum* of (1) the configuration specifications of the core cell and (2) the actual size of the padframe after all specified pads have been placed abutting one another in their respective positions. The type of pad placed in each position depends on the name indicated in the setup procedure.

To set parameters for padframe generation, click **Padframe Setup** in the **SPR Setup** dialog. L-Edit will display the **SPR Padframe Setup** dialog.

The dialog consists of two tabs—**General** and **Layout**. Each tab contains a **Reset** button, which will reset all fields and options to the values they held when you accessed the tab.

**SPR Padframe Setup—General**

This tab contains fields used to specify the cells and ports in the standard cell library that L-Edit will use for padframe generation.

Specify the following options:

- **Library cell names**
  - Specify the name of the **Power pad cell**, the **Ground pad cell**, and the **Corner pad cell** to be drawn from the standard cell library.
Port names

Names of the ports used to define the dimensions, positions, and orientation of the pad cells and padframe. These include the following:

- **Pad cell abutment port**—enter the name of the port used to define the edge of each pad cell.
- **Pad cell mirror port**—enter the name of the port used in the standard cell library to designate mirror ports in pad cells (for additional information, see “Mirror Ports” on page 1966). L-Edit will place pad cells with this port name in a special orientation.
- **Padframe port**—enter the name of the port used to define the inner edge of the padframe.

Output cell name

Specify the name of the generated **Padframe cell**.

**SPR Padframe Setup—Layout**

Use this tab to specify the padframe’s size and the location of the pads, plus individual characteristics of each pad in the padframe. **Initialize Setup** will complete this dialog automatically if your netlist contains pad connections.

Specify the following:

**Padframe dimensions (Display units)**  

**Width** and **Height** of the padframe. If you enter zero for any or both of the dimensions, L-Edit automatically determines the minimum size required.
The Padlist presents a numbered list of pads with their locations and attributes. Corner pads are not listed. If the padlist is empty, L-Edit will use the pad configuration in the netlist. The padlist contains the following columns:

<table>
<thead>
<tr>
<th>#</th>
<th>The number of the pad in the padframe. L-Edit orders pads counterclockwise along each side according to this number.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance name</td>
<td>The name of the pad instance. The name of the pad must be the same as the instance name in the netlist file. Pad cell names can be entered if no netlist is provided. (Padframe generation only.) See “Pad Cells” on page 1964 for naming conventions and restrictions.</td>
</tr>
<tr>
<td>Side</td>
<td>The side of the padframe on which the pad is placed. Pads must be entered into the pad list in the order left—bottom—right—top.</td>
</tr>
<tr>
<td>Mirror</td>
<td>Select either Yes or No. When you enter Yes for a given pad, L-Edit mirrors the pad through its vertical axis—unless this mirroring will conflict with mirroring information present in the cell library. For additional information, see “Mirroring” on page 1941.</td>
</tr>
<tr>
<td>Power/Ground</td>
<td>Enter Vdd and Gnd to designate particular pads as power and ground. Only one Vdd pad and one Gnd pad may be placed, and they must be on different sides.</td>
</tr>
</tbody>
</table>

To add an instance to the list, click Add. A New Pad is highlighted and can be edited. To delete a pad, highlight it by clicking any of its attributes and click Delete.

Adding Pads

L-Edit automatically places corner pads. If the padframe schematic contains fewer pads than the number required by the chip foundry for a complete padframe layout, you must complete the padframe by one of two methods:

- Adding placeholder pads at the appropriate locations in the padframe setup (for example, to have a total of 10 pads on each side of a 40-pin frame).
- Adding the required number of unconnected pad instances to the schematic, with module ports to specify their location.

Pad Naming and Ordering

Pad names entered in the padlist must meet one of the following criteria:

- The name must be exactly the same (except for case) as the instance name in the netlist file. For example, if the pad instance name is U21_3, enter U21_3 as the pad name.
- The name must match (except for case) the instance name in the netlist up to (but not including) the first occurrence of a < in the netlist instance name. For example, if the pad instance name is U21<1<333, enter U21 as the pad name.
The name must be exactly the same (except for case) as one of the pad cells in the standard cell library—for example, OPad or VddPad. Use this criterion when you only perform padframe generation; no netlist input is required in the SPR Setup dialog.

You can also specify pad information in the schematic (or directly in the netlist) by placing pads connected to the appropriate signals in the design. If the pads are to be in a specific order, then attach module ports to the PAD pins of each pad. (The PAD pin represents the location where a bonding wire will connect this signal to a pin on the chip.) These module port names must all be in one of the following formats. (In the following table, \( n \) is a number from 1 to the number of pads in the design, \( s \) is the first character of a side—L for left, R for right, and so on—and \( x \) is any string of characters.)

For information on defining pads in an EDIF netlist, see “EDIF Files” on page 1970.

<table>
<thead>
<tr>
<th>Format</th>
<th>Examples</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAD_( n )</td>
<td>PAD_1</td>
<td>The pads are placed in order. For example, PAD_1 is placed in the top of the left side of the padframe, PAD_2 just below it, and so on, traversing counterclockwise around the padframe. The more detailed form provides for port labeling. (See the figure “Pad order—Example 1” on page 1940.)</td>
</tr>
<tr>
<td>PAD_( n _x )</td>
<td>PAD_1_CLOCK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAD_2_DATA</td>
<td></td>
</tr>
<tr>
<td>PAD_( sn )</td>
<td>PAD_L1</td>
<td>The pads are placed in counterclockwise order on the given side. For example, PAD_L1 is placed at the top of the left side of the padframe, and PAD_L2 just below it. PAD_B1 is placed on the far left of the bottom of the padframe, and PAD_B2 just to the right of it. The more detailed form provides for port labeling. (See the figure “Pad order—Example 2” on page 1941.)</td>
</tr>
<tr>
<td></td>
<td>PAD_L2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAD_B1</td>
<td></td>
</tr>
<tr>
<td>PAD_( sn _x )</td>
<td>PAD_L1_CLOCK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAD_T3_ENABLE</td>
<td></td>
</tr>
</tbody>
</table>

Pad order—Example 1
Mirroring

Mirroring is used to mirror pads at specific locations. Incorrect mirroring could cause your chip to malfunction. Standard cell libraries provided by Tanner EDA contain information for automatic pad mirroring, if required. Where more specific mirroring information is needed, it is provided in the standard cell library file setup.

To use this information, select the TBD option from the Technology drop-down list in the Setup > Import Technology dialog. Select TDB as the type of file, select a TDB files from those in the drop-down list of open TDB files. In the SPR group, uncheck all options except for Padframe setup. (For placing mirroring information into your own pad cells, see “Pad Cells” on page 1964.)

**SPR Pad Route Setup**

The L-Edit pad router is a two-layer router. It first routes the power bus on one layer, then the signals on another layer. For each side of the padframe, there is a one-to-one correspondence between “connected” signals on the padframe and “connected” signals on the core. In other words, the uppermost “connected” signal on the left side of the padframe is routed to the uppermost “connected” signal on the left side of the core, the next “connected” signal down on the left side of the padframe is routed to the next “connected” signal down on the left side of the core, and so on. Each side of the padframe must have the same number of “connected” signals as there are on the corresponding side of the core.
To set parameters for pad routing, click **Pad Route Setup** in the **SPR Setup** dialog. L-Edit will display the **SPR Pad Route Setup** dialog.

The dialog consists of five tabs:

- **General**
- **Layers**
- **Design Rules**
- **Core Signals**
- **Padframe Signals**

Each tab contains a **Reset** button, which resets all fields and options to the values they held when you accessed the tab.
Chapter A2: Placing and Routing Standard Cells

SPR Pad Route Setup–General

In the **General** tab, enter the name of the **Chip cell**. The chip cell contains the core and the padframe instance, and it is where the pad routing will be placed.

SPR Pad Route Setup–Layers

The **Layers** tab is used to specify pad routing layers for I/O signals, power/ground signals, and pad vias (if needed).
Specify the following options:

**I/O signal layer**
Layer on which I/O signals are routed. In pad routing, I/O signals must be assigned to the same layer as that specified in “SPR Core Setup–I/O Signals” (page 1936).

**Power/ground signals**
Layer on which power/ground signals are routed. These must be routed on a different layer than I/O signals.

**Pad via layer**
Layer on which pad vias are drawn. L-Edit inserts pad vias if the ports of all pad cells are placed on a layer other than the I/O signal layer.

**SPR Pad Route Setup–Design Rules**

Use the Design Rules tab to specify design rule values L-Edit will use to perform pad routing that conforms to the fabrication technology used for your design. (The letters correspond to dimensions shown on pages 2-89 and 2-90.)

Values entered in this dialog are applied in the design as illustrated in the figures “Layer widths and spaces used in pad routing” and “Layer, core and pad spacing used in pad routing”, below.

The letters in the dialog fields provide a key for the labels in the following illustrations and the values they represent.
If the pad cell ports are on the pad routing layer, L-Edit applies the following design rules.

Layer widths and spaces used in pad routing

For I/O signals, all pad cell ports must be located on the same layer, but that layer need not coincide with the pad routing layer. In this case, L-Edit inserts a pad via, using the design rules illustrated below.

Design rules for pad via
**SPR Pad Route Setup—Core Signals**

Use the **Core Signals** tab to specify the signals entering or exiting the core. **Initialize Setup** will complete this dialog automatically if your netlist contains pad connections.

Enter the following information in the **I/O signal list**

**Signal name**
- Defines the names of all signals exiting or entering the core, beginning with the first signal on the upper left side of the core and proceeding counter-clockwise. Edit the **Signal name**, **Side**, or **Connect** (Yes or No) by selecting the item and typing the desired value.
- This list must contain as many signals as there are ports around the core.
- To add a signal to the list, click **Add**. A **New signal** is highlighted and can be edited. To delete a signal, highlight it (by clicking any of its attributes) then click **Delete**.

**Side**
- Side on which the signal is routed between the core and padframe. Options are:
  - Left
  - Bottom
  - Right
  - Top

**Connect**
- Options are **Yes** and **No**. To connect a signal to the padframe, type **Yes**; otherwise type **No**.
SPR Pad Route Setup—Padframe Signals

Use the Padframe Signals tab to specify a list of signals entering or exiting the padframe. Initialize Setup will complete this dialog automatically if your netlist contains pad connections.

Enter the following information in the I/O signal list:

**Signal name**
Defines the names of all signals exiting or entering the padframe, beginning with the first signal on the upper-left side of the padframe and proceeding counterclockwise.

This list must contain as many signals as there are pads around the padframe.

To add a signal to the list, click Add. A New signal is highlighted and can be edited. To delete a signal, highlight it by clicking any of its attributes then click Delete. To edit a signal, highlight it by clicking any of its attributes then click again to make the field editable.

**Side**
Side on which the signal is routed between the core and padframe.
Options are:
- Left
- Bottom
- Right
- Top

**Connect**
Options are Yes and No. To connect a signal to the padframe, type Yes; otherwise type No.
Standard Cell Place and Route

Use **Tools > SPR > Place and Route** to display the following dialog:
You can run the following three modules either in one step or separately:

- **Core place and route**—generates the design core using the options specified in “**SPR Core Setup**” (page 1926).
- **Padframe generation**—generates the padframe using the options specified in “**SPR Padframe Setup**” (page 1937).
- **Pad route**—routes between the padframe and the core using options specified in “**SPR Pad Route Setup**” (page 1941).

**Global input signal routing** allows you to perform a separate route of up to two input signal nets. This option is only available when you select **Core place and route**.

Clicking **Setup** opens the “**SPR Setup**” (page 1923) dialog.

Specify the following:

**Constraint**  
Specifies the factors used to constrain core size and shape. Options include:

- **Square**—Generates a core with four sides of approximately equal length.
- **Width**—Generates a core using the specified **Row length**.
- **Height**—Generates a core using the specified **Number of rows**.
- **Width and Height**—Generates a core using the specified **Row length** and **Number of rows**.

Selecting **Width and Height** will interrupt SPR after placement if the program cannot meet both constraints. In such a case, L-Edit will ask you whether you want to abort or continue the SPR run.

**Row length**  
Nominal length of rows placed in the core. Available only when **Constraint** is set to **Width** or **Width and Height**. This value is only an approximation, because the actual row length varies according to the number of row crosser cells inserted during routing.

**Number of rows**  
Number of rows in the finished core. Available only when **Constraint** is set to **Height** or **Width and Height**. Increasing the number of rows makes the core taller and thinner; decreasing it makes the core shorter and wider.

**Indent middle rows**  
Generates a core using the value entered in **Middle/top ratio**. This value is the ratio of the target length of the middle row of cells to that of the top and bottom rows.

Refer to “Indent Middle Rows” on page 1950 for guidelines on values for this field.

This value must be between 0 and 1 (inclusive); the default value is 1. Using 1 for the **Middle/top ratio** is equivalent to turning off the **Indent middle rows** option.

**Optimization** (in **Placement group**)  
Reduces core size by minimizing the overall netlength. When this option is off, L-Edit places cells according to their sequence in the netlist.
Indent Middle Rows

One factor affecting the width of standard cell rows is the number of row crosser cells inserted in a row. A row crosser is a small cell which contains a row crosser port but no logic; it simply provides a path for a signal to move through a row. When L-Edit needs to route a signal across a row of cells and no other cell in the row contains an unused row crosser port, the program inserts a row crosser cell.

Statistically, more row crossers are required in the middle rows of a design than in top or bottom rows, because the middle is more congested with logic. Hence, the middle rows might become significantly wider than they were estimated to be by the placement optimizer.

Select Indent middle rows for designs that have a significant number of row crosser cells added to their middle rows. If an initial SPR run produces a design whose middle rows are significantly wider than the top or bottom rows, use a Middle/top ratio of less than 1.
Placement Optimization

The core of a standard cell design contains rows of standard cells, which are designed to abut one another horizontally to form power and ground connections.

You can produce a more compact design by selecting Optimization in the Placement group. With Optimization on, L-Edit considers the positions and connections of standard cells and alters those positions where necessary to achieve a more compact layout. For example, if the output of a DFF is connected to the input of an inverter, the optimizer might relocate the DFF or the inverter to make the wire between them as short as possible.

With Optimization off, L-Edit simply places cells according to their sequence in the netlist.

When you select Optimization, you must also choose an Optimization factor. This factor enables you to control the degree of optimization and thus the quality of the placement. The higher the optimization value, the greater the total placement time will be. For additional information on this field, see “Optimization Factor,” below.

Optimization Factor

L-Edit uses a simulated annealing algorithm to optimize placement (described in Sechen, see “References” on page 1959). The optimizer algorithm randomly chooses pairs of cells and determines whether their locations should be exchanged in order to reduce the overall net length.

The Optimization factor is a measure of the number of states which the optimizer looks at per cell for every temperature step of the process, and so controls the placement time. For example, the placement for an optimization factor of 2 should be about an order of magnitude slower than a placement for an optimization factor of 0.2. Although it is possible to obtain better final results by increasing the factor by one or two orders of magnitude, a factor of 1 represents a balanced trade-off between total placement time and final core area. The following figure shows the average relationship between the final optimized core area and the optimization factor. Clearly, the improvement to the core area is not large for factors greater than 1.
Chapter A2: Placing and Routing Standard Cells

Output Options

Label Nodes

The **Label nodes with ports** switch instructs L-Edit to place ports on the layout (in the vertical routing layer) using the same port names as those used in the design schematic. You can use this option to extract a SPICE netlist with the original node names. This feature is also useful when it is necessary to perform any manual modifications to the results of the router, because it allows you to trace individual nodes as they wind through the core.

---

**Note:**

To minimize core size, it is best to run SPR several times with different optimization factors of around 1 rather than running it once with a single large optimization factor.

An optimization factor of 0 is not equivalent to “no optimization.” The optimizer will still run through placement, the row evener, global routing, and detailed routing with a minimal running time and a minimal effect on the placement optimization of the design. This is a good value to use while experimenting with other optimization controls (for example, changing the number of rows).

You can bypass placement optimization by clearing the **Optimization** check box. This is the fastest method for generating layout, but the final core will be significantly larger than a core produced with optimization.

**Note:**

To avoid excessive complexity in channel routing, it is recommended that you always turn **Placement Optimization** on for circuits with more than 2,000 standard cells.

---

The difference between the core area when optimization factor = 1 and the core area when optimization factor = 10 is within 5% in most cases.

The curve of larger circuits is slightly shifted to the right.

Optimization factor and core area for circuits up to 1000 standard cells.
Nodal Capacitance Files (CAP)

The option Write CAP file instructs L-Edit to compute the capacitance, length, and area added to each node due to routing. L-Edit writes the results to a plain-text file with the filename extension .cap.

Each line in the file is in the format:

node capacitance NoOfTerminals Length AreaOnH1 AreaOnV AreaOnH2

where node is the name of the node, capacitance is an integer denoting capacitance of this node in hundredths of a picofarad, NoOfTerminals is the number of pins attached to this node, and Length is the length of the interconnect of this node. AreaOnH1, AreaOnV, and AreaOnH2 denote the area of the route taken by this node on the H1, V, and H2 layers.

Note:

To use this file in a simulation, you must convert it to SPICE format.

Node capacitances are calculated based on the capacitance per unit area for a particular routing layer and the area occupied by the node. The capacitance per unit area for a layer is the capacitance between a particular routing layer and the substrate or the capacitance between two routing layers. Node capacitances also consider the fringe capacitance per unit length between the edges of the routing and the substrate.

The base values for capacitance between a layer and substrate or layer and another layer, as well as the fringe capacitance between the routing layer and the substrate, are entered by the user. These capacitance values are process-dependent and should be available from your chip foundry.

In the following discussion, we assume the horizontal routing layers to be Metal1 and Metal3 and the vertical routing layer to be Metal2. The capacitance on a node $C_{node}$ is computed as

$$C_{node} = C_{O,M12}A_{M12}^2 + C_{O,M23}A_{M23}^2 + C_{O,M31}A_{M31}^2 [\text{non-overlap}]$$

$$+ C_{O,M12}A_{M12}^2 + C_{O,M13}A_{M13}^2 [\text{overlap M1}]$$

$$+ C_{O,M21}A_{M21}^2 + C_{O,M23}A_{M23}^2 [\text{overlap M2}]$$

$$+ C_{O,M31}A_{M31}^2 + C_{O,M32}A_{M32}^2 [\text{overlap M3}]$$

$$+ C_{F,M1S}A_{M1S}^2 + C_{F,M2S}A_{M2S}^2 + C_{F,M3S}A_{M3S}^2 [\text{fringe}]$$

with

$$C_{O,M12} = C_{A,M12} + C_{A,M12}$$

$$C_{O,M13} = C_{A,M13} + C_{A,M13}$$

$$C_{O,M21} = C_{A,M21}$$

$$C_{O,M23} = C_{A,M23} + C_{A,M23}$$

$$C_{O,M31} = C_{A,M31}$$

$$C_{O,M32} = C_{A,M32}$$
where

$C_{A,\text{MXS}}$  
Area capacitance per unit area between the $MetalX$ layer and the substrate (entered using Setup > Layers—General).

$C_{A,\text{MXMY}}$  
Area capacitance per unit area between the $MetalX$ layer and the $MetalY$ layer (entered in the “SPR Core Setup—Layers” dialog).

$C_{O,\text{MYMX}}$  
Overlap capacitance per unit area on $MetalX$ when overlapped by routing on $MetalY$ (calculated by SPR).

$C_{F,\text{MXS}}$  
Fringe capacitance per unit length between the $MetalX$ layer and the substrate (entered using Setup > Layers—General).

$A_{\text{MX}}$  
Area covered by the route of this node in $MetalX$ with no overlap to any routing in any other layer (calculated by SPR).

$A_{\text{MXMY}}$  
Area covered by the route of this node in $MetalX$ that overlaps with routing in $MetalY$ layer (calculated by SPR).

$P_{\text{MX}}$  
Perimeter of all routing segments of this node in $MetalX$ (calculated by SPR).

Two-Layer Example

It is important to know how to extract numbers from the foundry’s actual process parameter sheet for entry in the dialog. In the following two-layer example, the process parameters are taken from a typical 2-micron $N$-well process. The value given for each layer (except for the last entry) is the area capacitance between the specified layer and the substrate, in aF/$\mu$m².

<table>
<thead>
<tr>
<th>Layer</th>
<th>$C_{A}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poly—Substrate</td>
<td>55</td>
</tr>
<tr>
<td>$N$ Diff—Substrate</td>
<td>128</td>
</tr>
<tr>
<td>$P$ Diff—Substrate</td>
<td>322</td>
</tr>
<tr>
<td>$Metal1$—Substrate ($C_{A,\text{M1S}}$)</td>
<td>25</td>
</tr>
<tr>
<td>$Metal2$—Substrate ($C_{A,\text{M2S}}$)</td>
<td>20</td>
</tr>
<tr>
<td>$Metal1$—$Metal2$ ($C_{A,\text{M1M2}}$)</td>
<td>38</td>
</tr>
</tbody>
</table>
Capacitance values are entered with two setup commands.

**Dialog**

<table>
<thead>
<tr>
<th>Setup Layers—General, Layer-to-substrate [Area] capacitance field</th>
<th>Values entered</th>
</tr>
</thead>
<tbody>
<tr>
<td>C&lt;sub&gt;A,M1S&lt;/sub&gt; = 25 and C&lt;sub&gt;A,M2S&lt;/sub&gt; = 20 are entered for Metal1 and Metal2, respectively.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPR Core Setup–Layers, Layer-to-layer area capacitance field</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C&lt;sub&gt;A,M1M2&lt;/sub&gt; = 38 is entered.</td>
<td></td>
</tr>
</tbody>
</table>

The overlap capacitances of Metal1 and Metal2 are internally calculated as follows:

- An overlapping node on Metal1 sees an overlap capacitance of C<sub>A,M1S</sub> + C<sub>A,M1M2</sub>.
- An overlapping node on Metal2 sees an overlap capacitance of C<sub>A,M1M2</sub> only.
- Thus, C<sub>O,M1M2</sub> = C<sub>A,M1S</sub> + C<sub>A,M1M2</sub> = 63 and C<sub>O,M2M1</sub> = C<sub>A,M1M2</sub> = 38.

**Standard Delay Format Files (SDF)**

When you run SPR with the option **Write SDF file**, L-Edit computes the delays due to routing and outputs the results into a standard delay format (SDF) file with the filename extension `.sdf`.

The L-Edit interconnect delay calculation is based on the Elmore delay model. The capacitance and resistance is extracted segment by segment and distributed as a π model.
Each segment of the interconnect is associated with a lumped $R$ and $C$ value.

![Diagram of interconnect segments](image)

The $R$ and $C$ values are determined by

$$R = R_{square} \cdot \text{(Length of segment} / \text{Width of segment)}$$ (0.8)

$$C = C_{area} \cdot \text{Area of segment} + C_{fringe} \cdot \text{Perimeter of segment}$$ (0.9)

where width of segment is defined as the wire edge that connects to the pin and

$$\text{Perimeter} = 2 \cdot (\text{Length of segment} + \text{Width of segment})$$ (0.10)

**Note:** Capacitance with respect to the substrate is the only capacitance component that is considered. No attempt is made to include layer-to-layer or crosstalk capacitance terms.

The detailed $R$ and $C$ values are calculated as follows:

For a segment located on Metal$n$:

$$R = R_{Mn} \cdot \text{Number of squares} = R_{Mn} \cdot (L_{seg} / W_{seg})$$ (0.11)

$$C = C_{A,MnS} \cdot A_{Mn} + C_{F,MnS} \cdot P_{Mn}$$ (0.12)

where

- $R_{Mn}$: Resistivity (resistance per square) of the Metal$n$ layer
  - Entered in Setup Layers—General.
- $C_{A,MnS}$: Area capacitance per unit area between the Metal$n$ layer and the substrate
  - Entered in Setup Layers—General.
- $C_{F,MnS}$: Fringe capacitance per unit length between the Metal$n$ layer and the substrate
  - Entered in Setup Layers—General.
Chapter A2: Placing and Routing Standard Cells

Output Options

Note:
The area and perimeter term includes the small amount of overlap between layers. The error resulting from this inclusion is negligible.

The \( R \) and \( C \) values for each segment are distributed as a \( \pi \) model. For the above interconnection, the \( R \) and \( C \) values are distributed as follows:

\[
\text{Pin-to-Pin Delay Calculation}
\]

The pin-to-pin delay is the interconnection delay between a driver pin and a receiver pin. Driver pins are all output or bidirectional pins on a net; input pins are all input or bidirectional pins on a net. In this example, the pin \( C \) is the driver pin and \( A \) and \( B \) are the receiver pins.

\[
\text{Delay(Pin C to Pin A)} = R_{\text{OUT},C} \times (C_5 + C_4 + C_3 + C_2 + C_1 + C_{\text{IN},A} + C_{\text{IN},B}) + R_5 \times (0.5C_5 + C_4 + C_3 + C_2 + C_1 + C_{\text{IN},A} + C_{\text{IN},B}) + R_4 \times (0.5C_4 + C_3 + C_2 + C_1 + C_{\text{IN},A} + C_{\text{IN},B}) + R_3 \times (0.5C_3 + C_2 + C_1 + C_{\text{IN},A} + C_{\text{IN},B}) + R_2 \times (0.5C_2 + C_1 + C_{\text{IN},A}) + R_1 \times (0.5C_1 + C_{\text{IN},A})
\]

This delay calculation corresponds to the 63.2% threshold voltage of the single-pole response.

The SDF computed by L-Edit contains interconnect delays for both rising and falling edges. These are computed using different values for the driver impedance \( R_{\text{OUT}} \). The values for \( R_{\text{OUT}}, C_{\text{IN}}, \) and \( C_{\text{OUT}} \) (if
any) are obtained from properties placed on the corresponding ports of the standard cell layout used by SPR. These properties may be modified in two ways:

- You can enter them using the **Tools > Add-Ins > SDF Driver Properties > “Edit Pin Characteristics”** (page 1959). The resistance values are given in ohms, and capacitance values are in Farads.
- You can import the data from a Liberty timing (.lib) file using **Tools > Add-Ins > SDF Driver Properties > “Import .LIB Timing Data”** (page 1958). In this case, all cells found in the library file are annotated. A fragment of a .lib file is shown below:

```plaintext
library (tsmm025DL) {
    pulling_resistance_unit : "1kohm";
    capacitive_load_unit (1.0, ff);

    cell (Buf1) {
        pin (A) {
            direction : input;
            capacitance : 0.983;
        }
        pin (OUT) {
            direction : output
            timing() {
                rise_resistance : 3.70;
                fall_resistance : 3.63;
            }
        }
    }
}
```

### SDF Driver Properties

SDF Driver Properties is a macro that allows you to specify driver/receiver data by importing a .lib file or by editing driver/receiver properties on specific ports. SPR uses the driver/receiver properties on the zero-height ports in the standard cell for SDF calculations. (See the Application Notes for additional details.)

### Import .LIB Timing Data

You can import timing data for SDF from a Liberty Timing File (.lib). To import a .lib file, select **Tools > SPR > SDF Driver Properties > Import .LIB Timing Data...** from the L-Edit menu.

Type the name of the appropriate .lib file, or click **Browse** to open a standard file browser. Click import to set the port properties specified in the .lib file.
Edit Pin Characteristics

To edit driver/receiver characteristics on a selected port, choose the menu option **Tools > SPR > SDF Driver Properties > Edit Pin Characteristics...**

![Edit pin characteristics](image)

Using the pull-down menu, choose a pin **Type** of **Input**, **Output**, or **None**.

If the pin type is **Input** or **Output**, you can edit pin characteristics in the appropriate fields. Options include:

- **Load capacitance** (Input and Output pins.)
  - The load capacitance of the selected pin.

- **Pull-up resistance** (Output only.)
  - The effective drive impedance of the logic gate when it is trying to drive a logic “1” (high voltage).

- **Pull-down resistance** (Output only.)
  - The effective drive impedance of the logic gate when it is trying to drive a logic “0” (low voltage).

Pull-up and pull-down resistance are usually determined experimentally or by simulation, by measuring the propagation delay of a given gate layout driving various values of load capacitance. The slope of this curve is the drive impedance (pull-up or pull-down), while the delay for C=0 gives the intrinsic gate delay (also known as the internal gate delay).

**References**

Standard Cell Library

The cells in a standard cell library must meet certain constraints of dimension and port positions for proper use by L-Edit. Usually, a standard cell library includes two types of cells:

- **Standard** cells, which L-Edit can place and route.
- **Pad** cells, an optional set, which L-Edit uses in padframe generation and routing.

Standard Cells

Abutment Ports

Each standard cell should have a special abutment port whose name is consistent with the Abutment port entry in the “SPR Core Setup-General” (page 1927) dialog. The dimensions and position of an abutment port correspond to the boundaries of the cell to which it belongs. The abutment port must have the same height in all standard cells in a library set. Abutment port widths should also be integer multiples of the vertical routing pitch.

*Note:* Channel routing will take less time when the abutment ports are of a uniform width and spacing.

Power Ports

Power buses enter and exit at the ends of the standard cell rows, run horizontally along the row, and connect to power ports within each cell, which must be placed on both sides of a standard cell at the cell boundary. Power port names are specified in the Power Signal and Ground Signal fields in the SPR Setup dialog.

Ports for a power terminal (Vdd or Gnd) must have the same height and position relative to the abutment port in every standard cell of a library set. The width of power ports has to be zero. The power
rail can run along the left or right side of the core, as specified in the “SPR Core Setup–I/O Signals” (page 1936) dialog.

![Typical power bus arrangement and power port positions.](image)

**Signal Ports**

Signals other than power and ground are routed to ports through the top or bottom sides of a standard cell. A signal port must have a height of zero and a name complying with that of standard cell primitives in the netlist, and it must be placed on the same layer as the vertical routing wires.

Signal ports for signal routings must be placed at related layout geometry where the signals are available. Signal port positions must comply with the relevant design rules (see “SPR Core Setup–Design Rules” (page 1930)), with predefined routing width and space specifications. Signal ports can be either inside the standard cell or on its boundary (see the illustration “Signal port positions in standard cells” on page 1962.)

**Note:** Channel routing will take less time when the signal ports are of a uniform width and spacing. Signal port widths should also be integer values.
There are three options for routing wires to fit specific layout features of a standard cell set.

**Row Crosser Ports**

To route wires between two routing channels—that is, across a standard cell row—L-Edit uses user-specified row crosser ports that identify crossing paths on standard cell rows. (Row crosser ports illustrated in figures in this chapter have the port name Cross.) The use of row crosser ports is illustrated in a regular standard cell in the figure “Signal port positions in standard cells” on page 1962 and in a dedicated row cross cell in the figure “Routing wire arrangements in standard cells” on page 1962.
In a regular standard cell, it is a good practice to place as many row crosser ports as design rules and SPR constraints allow. This helps L-Edit increase area efficiency, because if there are no more row crosser ports in standard cells within a certain row span, L-Edit may have to insert a row crosser cell.

A row crosser cell is a standard cell that contains only one row crosser port and is placed only to make up a cross-row pass. In the figure “Signal port positions in standard cells” on page 1962 (b), L-Edit will treat the pairs of port Cross as a crossing pass between the routing channels above and below the current standard cell row. As with signal ports, it is assumed that the pair are internally connected with layout geometry in related layers in the standard cell. L-Edit picks pairs of row crosser ports from left to right. Dotted lines in the figure connect related upper and lower signal ports. The figure “SPR box generation for design rule correctness of a row crosser port” on page 1963 illustrates how L-Edit automatically generates extra geometry around a row crosser port to ensure design rule correctness at that location.

**Note:** Row crosser ports must be placed on the vertical routing layer. SPR will not recognize such a port when it is placed on the horizontal routing layer.

### Special Standard Cells

L-Edit requires three special standard cells to be included in a library cell set. These cells are not standard cell primitives like the ones included in the netlist; they are for node connections only.

The *Tie-to-Power Cell* and *Tie-to-Ground Cell* are needed where a standard cell has a pin directly tied to Vdd or Gnd. The *Row Crosser Cell* is a special standard cell that contains only a row crosser port. Its sole purpose is to allow a connection between two channels located above and below a standard cell row. The figure “Connection cells in a standard cell set,” below illustrates these three typical connection cells.
Although a given SPR operation might not require these three cells—the use of the Tie-to-Power cell or Tie-to-Ground cell depends on the specific netlist, and the use of row crosser cells depends on the actual routing condition—L-Edit treats them as prerequisite and elementary parts of the standard cell library. Specify the names of the Tie-to-Power, Tie-to-Ground, and row crosser cells in the “SPR Core Setup–General” (page 1927) dialog. L-Edit will report an error if any of these three cells are missing in the standard cell library.

Pad Cells

Abutment Ports

Each pad cell should have a special abutment port whose name is consistent with the entry Pad cell **abutment port** in the “SPR Padframe Setup–General” (page 1937) dialog. The dimensions of an abutment port geometry specify the boundaries of the cell to which it belongs. The abutment port must have the same height in all pad cells in a library set.

Connection Ports Between Pad Cells

When pad cells are placed in a padframe, power buses and signal buses (if specified) run horizontally across each pad cell. During padframe generation, L-Edit places pads in a user-specified padframe and fills the gaps between pads with cell connections to assure continuity of power and/or signal buses to all pads. These interpad connections must have dedicated ports on two sides of the pad cell boundary. Power ports in all pad cells in a library set must have the same height and the same position along the cell boundary. The existence of interpad connection ports in a particular layer specifies a connection in that layer.
The figure “Typical pad cells lined up in a segment of a padframe” on page 1965 (simplified so that only straight signal passes and power buses are shown) illustrates connections between pad cells A and B. In padframe generation, L-Edit can optimize the padframe under certain conditions so that adjacent pads are attached to each other, as shown between pad cells B and C. In this case, the abutment ports (labeled Abut) specify the pad cell boundaries and allow L-Edit to abut and align pad cells. The same figure also shows typical arrangements of power bus ports (on the sides) and signal ports (on the bottoms) in pad cells.

Signals from Pad to Layout Core

In order to interface with the layout core of a chip, signal ports must be available on the bottom boundary of a pad cell. A signal port is a zero-height port with its width equal to the layout path of the dedicated signal.

Power Supply Pads

In padframe generation, L-Edit automatically places one Vdd pad cell and one Gnd pad cell into the frame. Their cell names can be customized in the “SPR Padframe Setup” (page 1937) dialog. A power supply pad cell is subject to the same structure constraints as a normal signal pad. There must be at least one pair of power supply pads in a padframe, which provide both power connections to all pads in the padframe and to the layout core of the chip. It is not necessary to specify these power supply pad cells as library primitives in your netlist. If your design requires secondary power supply pads, specify them in the netlist as you would regular signal pads.

Corner Pad Cells

Standard cell libraries must include a special corner pad cell, which is required to complete a padframe. This cell will be oriented and placed at all four corners of the padframe. A corner pad usually contains no active circuit, signal path, or bounding area because it does not directly face the layout core. Corner cells continue the power bus and other signal connections between two perpendicular sides of the
padframe. They may also contain certain electrostatic discharge (ESD) protection guardbands in structures as they are built in regular pad cells. For proper extension of power buses and signals, corner pad cells must meet the same design requirements as regular pad cells. In particular, they must contain side ports like those created in regular pad cells.

**Pad Cells Without Bond Pads**

In some cases, a pad cell is needed to fill a gap in a segment of a padframe. Pad cell sets are allowed to have pad cells without bonding pads. Such a pad will not be bondable, but it can serve as a padframe spacer cell. L-Edit allows any such unbondable pad cells to occupy a pad slot in the padframe specification, as specified in the “**SPR Padframe Setup**” (page 1937) dialog. The figure “Mirror-labeled pads in a padframe” on page 1967 illustrates how pad slots are indexed in the **SPR Padframe Setup—Layout** dialog. Typical uses of such spacers are as padframe corner cells and padframe spacer cells. L-Edit requires corner cells to complete a padframe; spacer cells may be useful optional cells when specific padframe geometry and dimensions are required by a chosen process vendor. Since no signal paths lead to the layout core of the chip, it is not necessary to include these types of pad cells in the netlist primitive set.

**Pad Orientations**

The **Mirror** switch in the **SPR Padframe Setup—Layout** dialog, set by typing Yes or No in the **Mirror** column of the padlist, instructs L-Edit to mirror an individual pad when placing it in a padframe. This feature is especially useful when a pad cell contains asymmetrical features that would affect intercell connections. In the figure “Using a mirrored pad cell in a padframe,” below, pad cell A has a wider ground bus on the left side and wider power bus on the right. Consequently, a power short exists between the first pad cell A on the left and its duplicate on the right—i.e., Vdd in the left pad cell A has been connected to Gnd in the duplicate pad cell A on the right. The power short has been avoided in the next two duplicated pairs of pad cell A, because the third pad cell A has been placed in its mirrored orientation.

**Mirror Ports**

You can specify a **Pad cell mirror port** in the “**SPR Padframe Setup—General**” (page 1937) dialog. When L-Edit encounters a pad cell with the specified mirror port name, the program automatically alters this pad cell’s orientation when placing it adjacent to another pad cell labeled as a mirror port. In
addition, this mirroring feature can be propagated through a padframe’s corner pad cell if the corner pad cell has also been labeled with a mirror port.

The figure “Mirror-labeled pads in a padframe” on page 1967 shows an example of an SPR-generated padframe with some typical mirroring effects. All pad cells and padframe corner cells contain mirror ports. Pad cells labeled m are in mirrored orientation, while other cells are on their normal orientation.

Designing Cells for Global Signal Routing

Global Signal Port Definitions

For global input signal routing, standard cells and buffer cells are extended by two global signal buses, which are located above and below the power buses on the horizontal layer. Each bus contains two ports (labeled GS1 and GS2 in the following figure). These port names are defined during cell design, and they thus become part of the cell definition in the standard cell library.

Global signal port names are assigned to the global input signals in the “SPR Core Setup–Global Signals” (page 1934) dialog. Note that in this dialog, SPR users also assign a specific net name to each of the global input signals. Thus, the assignment of a global input signal port to each of the global input signals determines which signal bus (upper or lower) represents a specific global signal net. Because
buffer cells are specifically dedicated to either the upper or lower global signal bus, this port assignment also determines:

- Which global signal rail (left or right) represents a specific global signal net.
- The side of the padframe on which the pad of this net is placed.

As an example, let NetA be the name of a signal designated Global Input Signal 1. Assume that the standard cell library contains buffer cells designed as those shown in the figure “Global signal ports (GS1 and GS2) in standard cells and buffer cells” on page 1967, with the left buffer cell connecting to the upper global signal bus, which is labeled with the port GS1. If port GS1 is designated as the port for Global Input Signal 1 (which is NetA), the following will occur:

- In the standard cells, the internal signal port for NetA will be connected to the upper global signal bus because GS1 is its assigned port name.
- Because the left-side buffer cell is designed such that it drives the upper global signal bus, the left global signal rail will carry NetA.
- Subsequently, it will be necessary to specify that NetA exit the left padframe edge (see “SPR Pad Route Setup–Padframe Signals” (page 1947)).

**Note:**

Because buffer cells are specifically dedicated to either the upper or lower global signal bus, buffer cell design determines the relationship between port names and the side on which the net associated with this port leaves the core. The buffer cell designer thus determines the side of the padframe to which global signal ports are ultimately connected.

### Buffer Cell Input Ports

Besides meeting all general design constraints imposed on standard cells, buffer cells used in global input signal routing must also meet the following constraints:

- Each buffer cell must contain an input port located on one side of its abutment port. The left buffer cell has its input port on the left side of its abutment port; the right buffer cell has its input port on the right side of its abutment port.
- Input ports on buffer cells must be vertical ports (height > 0, width = 0) and reside on the vertical routing layer.

The height of the input port determines the width of the global signal rail connected with it. The width of the global signal rail will be twice the height of the input port. Buffer cell input port names can be arbitrary.
A4 Place and Route File Formats

L-Edit standard cell place and route accepts Tanner Place and Route (TPR) and Electronic Design Interchange Format (EDIF) input files, and generates Standard Delay Format (SDF) and Nodal Capacitance files (CAP) output files.

TPR Files

L-Edit can use netlist files in Tanner Place and Route (.tpr) format to generate chip layouts. TPR files are ASCII text files that are generated automatically by the schematic editor S-Edit; they can also be created with any text editor.

Syntax

A portion of the .tpr netlist file for the bargraph example is shown below.

```
Comment line
$ TPR written by the Tanner Research schematic editor, S-Edit
$ Version: 2.0 Beta 5     Jan 7, 1998  16:07:16

Pad cell definition
CP PadOut DataOut Pad;
UPadOut_1 N2 PAD_B1_131;

Instance definition
CP PadInC DataIn DataInB DataInUnBuf Pad;
UPadInC_1 N68 IPAD_9/N2 IPAD_9/N1 PAD_L9_SCO;

Ground pad
CP PadGnd Pad;
UPadGnd_1 PAD_R8_GND;

Power pad
CP PadVdd Pad;
UPadVdd_1 PAD_L6_VDD;

Cell definition
C INV A Out;
UINV_3 BARGRAPH_1/BG64_2/N9 BARGRAPH_1/BG64_2/SFT3;

Instance definition
C Mux2 A B Out Sel;
UMux2_1 BARGRAPH_1/BG64_1/BG4_1/N118 BARGRAPH_1/BG64_1/BG4_1/N108
N62 + BARGRAPH_1/BG64_1/S11;
```

In the two lines above, DataIn, DataInB, and DataInUnBuf are the names of ports in the pad cell PadInC (PortList). N68, IPAD_9/N2, and IPAD_9/N1 are the names of nets attached to these ports (NetList). PAD_L9_SCO is the name given to the body region of the pad. L9 identifies the position of the pad as the ninth pad from the top on the left side of the padframe.
In the three lines above, A, B, Out, and Sel are ports in the standard cell Mux2 (PortList). BARGRAPH_1/BG64_1/BG4_1/N118, BARGRAPH_1/BG64_1/BG4_1/N108, N62, and BARGRAPH_1/BG64_1/S11 are the names of nets attached to these ports (NetList). Note that these net names include the hierarchical structure of the schematic. This is the manner in which S-Edit creates a “flattened” .tpr netlist. A plus sign (+) indicates a continuation of the previous line.

**Interpretation**

Pad cells are defined in the format:

```
CP <padname> <pin1> <pin2> ... Pad
U<gateUID> <net1> <net2> ... Pad_<PadPosition>
```

Standard cells are defined in the format:

```
C <cellname> <pin1> <pin2> ...
U<gateUID> <net1> <net2> ...
```

A .tpr file must conform to the following rules:

- All signals which are to be routed within the core or from the core to the padframe are required to be listed, with the exception of the Vdd and Gnd signal connections to pads.
- For each cell, the PortList and NetList must have the same number of elements.
- The name “PAD” in the PortList of a pad cell refers to the actual bonding region of the pad, and is not actually involved in the placement and routing process. Pad cells must have a signal marked “PAD.”
- The bonding region of a pad can contain the location of the pad on the padframe. For example, “B1” stands for the leftmost pad on the bottom side of the padframe (L = Left, B = Bottom, R = Right, T = Top).
- Power and ground pads do not have to be included in the netlist. If they are not included, SPR will place them automatically.
- The parts listed in the file must match the cells contained in the layout library. To match, the name of the part must be identical to the name of the library cell (except for case), and every signal listed in the part description must have at least one port of the same name somewhere in the library cell.

**EDIF Files**

Netlist files in Electronic Design Interchange Format (EDIF) are used by SPR to place and route a design. EDIF netlist files typically have a filename extension of .edf, .edn, or .edi.

SPR requires flattened EDIF netlists or netlists with one level of hierarchy.

L-Edit supports flattened EDIF version 2 0 0 with EDIF level 0, keyword level 0, and netlist view—(edifLevel 0), (keywordLevel 0), and (viewType NETLIST) files. Other view types are ignored.

The netlist parser is limited to one netlist view label per netlist. If the netlist contains more than one netlist view, L-Edit warns you and ignores subsequent, different view labels.

All cell properties will be transferred to the relating instance. Properties with no relation to a netlist view will be ignored. L-Edit currently supports the use of properties with regard to the labeling and positioning of pads and I/O signals. If an EDIF netlist contains both pad properties and I/O signals, only
the pads will be considered. L-Edit provides an optional warning when both appear in a netlist, which may be disabled using Setup > Application > Warnings.

The parser is limited to one design—(design designname (...))—per EDIF netlist file.

External EDIF library definitions—(external libraryname (...))—are treated in the same way as normal EDIF library definitions—(library libraryname (...)). Furthermore, the parser considers only library information present in the current EDIF netlist file.

A mapping table is generated if cell names or port names in the EDIF netlist differ from those used in the cell library. L-Edit automatically accesses the Mapping Table dialog whenever it encounters a cell or port name discrepancy between the EDIF netlist and the standard cell library. You can also access this dialog via the SPR Setup dialog. The mapping information is stored in the TDB design file.

**Syntax**

The following example shows excerpts from an EDIF netlist containing pad cells, I/O signals and critical nets. If pad cells and I/O signals are both included in the netlist, the pad cell configuration has precedence over I/O signals.

```plaintext
(edif bargraph
  (edifVersion 2 0 0)
  (edifLevel 0)
  (keywordMap {keywordLevel 0})
  (status
    (written
      (timestamp 1998 11 01 07 5 00)
      (program "S-Edit" (version "Version 2.06")))
    (library bargraph_top
      (edifLevel 0)
      (technology (numberDefinition (scale 1 (E 1 -12) (unit CAPACITANCE)))
        (cell bargraph_top
          (cellType GENERIC)
          (status
            (view view_1
                (viewType NETLIST
                    (interface
                      (port ClB (comment "I/O Signal")
                        (property PIN_LOCATION (string "L2")
                            (direction INPUT))
                      (instance PadInC_1
                          (viewRef view_1 (cellRef PadInC))
                          (portInstance DataIn)
                          (portInstance DataInB)
                          (property PAD (string "L1"); (comment "Pad")
                      )
                      (net N54
                          (joined
                            (portRef DataInB (instanceRef PadInC_5))
                            (criticality 100) (comment "Net criticality")
                          )
                      )
                      (net N55
                          (joined
                            (portRef ClB) (comment "Reference to an I/O Signal")
                            (criticality 100) (comment "Net criticality")
                          )
                      )
                    )
                )
            )))
        )))
  )))
)
```


Chapter A4: Place and Route File Formats

EDIF Files

)
))
)
(design ROOT
(cellRef bargraph_top
(libraryRef bargraph_top)))
)

Interpretation: Pads
Pad cells are defined by creating a property named PAD with a value such as L1 in the pad cell instance.
The following formats are supported:
(property PAD (string "L1"))

or
(property PAD (string "1"))

The string value determines the position of the pad, counting counterclockwise. (In this example, the
pad is placed on the upper-left position of the padframe.) In the first format, the sides of the padframe
are labeled with L (left), B (bottom), R (right), and T (top). The subsequent number determines the
position on this side. The second format labels the pad position with only a numeric value ( > 0). The
resulting position is determined according to this value, counting counterclockwise, starting from the
upper left position on the padframe. In other words, pad position L1 is equal to pad position 1. To avoid
ambiguity, it is recommended that only one format be used in the same netlist file.
Pads with no string value attached, e.g., (property PAD (string "")), are equally distributed around the
padframe.

Interpretation: I/O Signals
If you intend to generate an SPR core cell only, you don't need to specify pad cells. In this case, ports
assigned to the top-level cell in your EDIF netlist can be assigned to I/O signals leaving the core. Please
note that I/O signals will only be considered if the netlist does not contain pad cells. (If both are present,
pad cells will be considered and I/O signals will be ignored.)
I/O signals are defined through the EDIF interface keyword. When SPR reads the netlist, it uses the
I/O signals designated in the interface section to initialize the core I/O signals setup (SPR > Setup >
Core Setup—I/O Signals dialog). You can change the signal location in this dialog to suit your design.
In most schematic editors, when you define pins or ports to a schematic or symbol, they are designated
as I/O signals. The following is an example showing the I/O signal CLK:
(interface
(port CLK
(direction INPUT)
)

In the above example, no pin position has been provided. In this case, the signal CLK will be equally
distributed with the other I/O signals around the core during placement. (A specific I/O signal position
can be defined in the (SPR Core Setup—I/O Signals dialog.)
Alternatively, you can also provide an I/O signal port position as shown below:
(interface
(port CLK

L-Edit 16 User Guide

Section 2

1972


This example will place the I/O signal \texttt{CLK} on the second position (top to bottom) on the left side of the core. The EDIF property \texttt{PIN\_LOCATION} with the property value \texttt{1, 2, ...} or \texttt{L1, L2, ..., B1, ..., R1, ..., T1...} indicates the relative position in which the I/O signals will be placed around the core, counting counter-clockwise.

\textbf{Interpretation: Criticality}

Critical nets are defined by using the \texttt{criticality} construct in EDIF. SPR considers critical nets during placement optimization. Criticality is expressed as an integer value. It may be positive (this net is given a higher priority for placement purposes) or negative (this net is given a lower priority during placement). The default value for any net that is not specified with a criticality value is zero.

The consideration of net criticality in SPR is based on two assumptions:

- The numerical value of the criticality describes the relative importance of a net compared to others. For example, if the criticality of net A is twice the criticality assigned to net B, then the placer considers it twice as important to reduce the length of net A compared to net B.
- The critical values are scaled internally according to the largest value that has been entered, with the largest value assigned to a fixed internal value. For example, if net A is the only net with an assigned criticality, then any criticality value greater than zero for this net would lead to the same result.

\textbf{Additional Notes}

An EDIF netlist file must conform to the following rules:

- An EDIF netlist component (e.g., cell, port) must be defined completely before it can be used.
- If your cell interface in the EDIF file contains ports which are not connected in your design, you can label them as “not used” during the mapping process.
- All signals that are to be routed within the core or from the core to the padframe must be listed, with the exception of VDD and GND signal connections to pads.
- Power and ground pads do not have to be included in the netlist. If they are not included, L-Edit places them automatically in accordance with the power and ground rails.
- The range of integer numbers is \(-2^{31}+1 \leq x \leq 2^{31}-1\) (32-bit signed integers). Real numbers are valid in a range of \(-1 \times 10^{35} \leq y \leq 1 \times 10^{35}\). The length of a string is limited to 256 characters. The length of a line is limited to 512 characters.
- The array construct \((\text{array } \text{arrayname} (...)\)) in a name definition is limited to one- or two-dimensional arrays.
- Valid EDIF identifiers consist of alphanumeric or underscore characters, and must be preceded by an ampersand (\&) if the first character is not alphabetic. Thus, “pure” integer numbers are not allowed as identifiers.
- An ampersand (\&) at the beginning of an identifier will be ignored. The case of a character is not significant. For example, \&Nand2, \texttt{Nand2}, and \texttt{nand2} all represent the same EDIF name.
References


SDF Files

Pin-to-Pin Delay Syntax

```plaintext
(DELAYFILE
 (SDFVERSION "OVI Standard 3.0")
 (DESIGN "bargraph")
 (DATE "02/22/1999")
 (VENDOR "Tanner Research, Inc.")
 (PROGRAM "L-Edit/SPR")
 (VERSION "8.0")
 (DIVIDER /)
 (VOLTAGE)
 (PROCESS)
 (TEMPERATURE)
 (TIMESCALE 1ps)
 (CELL
  (CELLTYPE "bargraph")
  (INSTANCE bargraph)
  (DELAY
   (ABSOLUTE
    (INTERCONNECT NAND_1/C NOR_2/A (0.005))
    (INTERCONNECT NAND_1/C NOR_2/B (0.003))
    etc.
   )
  )
 )
 )
)
```

Interpretation

The pin-to-pin delay is generated in the OVI SDF Specification Standard 3.0:

```
INTERCONNECT <port_instance_1> <port_instance_2> (<delay>)
```

*port_instance_1* is an output or bi-directional port. *port_instance_2* is an input or bi-directional port. *delay* is the interconnect delay between the output and the input ports.

The **DESIGN** entry in the SDF file header indicates the name of the design—that is, the name of the TDB file.

The **CELLTYPE** entry indicates the name of the cell—either the chip cell name (if pad route is included) or the core cell name (if core route only).
CAP Files

L-Edit standard cell place and route calculates the nodal capacitances and other characteristics of the interconnect and outputs this information to a nodal capacitance file with the extension .cap.

Syntax

The following is an example of a CAP file.

```plaintext
$ Nodal Capacitance File: D:\ledit_files\v8_shipping\example3\bargraph.cap
$ SPR Date and Time: 02/27/1999 - 8:00
$ ...
$ H1 layer-to-substrate cap. - Area: 36 aF/sq.micron Fringe: 0.086 fF/micron
$ V layer-to-substrate cap. - Area: 11 aF/sq.micron Fringe: 0.077 fF/micron
$ H2 layer-to-substrate cap. - Area: 7 aF/sq.micron Fringe: 0.031 fF/micron
$...
$ 1 Locator Unit (LU) = 1/1 Lambda = 7/20 Micron(s)
$...
$ Node Capacitance No of Terminals Length Area On H1 Area On V Area On H2
$ (1/100 pF) (LU) (LU^2) (LU^2) (LU^2)
$...
N4 6 9 5964.40678.00010567.20013296.000
N27 7 2 6046.000975.00012480.0009366.000
N26 2 2 2452.0000.0004479.0005754.000
N25 9 2 7589.0000.0001829.00011829.00017130.000
N24 5 2 4062.0000.0003936.00016500.000
N12 1 2 599.5000.0001798.5000.0000
$...
$ Length of all nets (LU): 1893584.90
```

Interpretation

Each line in the file is in the format:

```
node capacitance NoOfTerminals Length AreaOnH1 AreaOnV AreaOnH2
```

where:

- **node**
  - Name of the node.

- **capacitance**
  - An integer denoting capacitance of this node in hundredth of a picofarad.

- **NoOfTerminals**
  - Number of pins attached to the node.

- **Length**
  - Length of the interconnect of the node.

- **AreaOnH1**
  - Area of the route taken by the node on the H1 layer.

- **AreaOnV**
  - Area of the route taken by this node on the H2 layer.
AreaOnH2

Area of the route taken by this node on the H2 layer.

For a detailed description of how nodal capacitances are calculated, see “Output Options” on page 1952.
Netlist Extraction

Legacy Layer Generation

When you use generated layers in an extract definition, L-Edit automatically generates objects on those layers before proceeding with netlist extraction. Following netlist extraction, L-Edit automatically deletes the objects it created during that Extract run. However, it only deletes objects generated during that most recent Extract run—previously generated objects remain.

The extractor uses the layer generation method found in L-Edit version 9, which only operates on boxes, and 45° and 90° polygons and wires. It does not extract circles or all-angle polygons and wires.

This behavior is different from layer generation using Tools > DRC or Tools > Generate Layers. If you need to manually run the Extract layer generation, you can do this using “Legacy Layer Generation,” below.

Legacy Layer Generation

Because the extractor generates layers automatically, you do not need to perform layer generation as a separate step. However, if you wish to manually generate layers to see them as the extractor sees them, you can do so using Tools > Add-Ins > Legacy Layer Generation:

- **Target Layer(s) to be generated. Options include:**
  - All layers
  - Layer—a single layer, selected from the drop-down menu

If a derived layer is dependent on source layers that are themselves derived, the source layers are recursively generated as well.

**Note:** L-Edit does not generate locked source layers or layers for which derivation is disabled. In these cases, L-Edit treats the source layers as drawn layers for the purposes of generating the derived layer.
| **Delete all derived layers prior to generation** | Clears objects on all derived layers. When generating a single Layer, use this option to clear objects on other derived layers. Ports on derived layers are not deleted. |
| **Binning** | When checked, L-Edit divides the layout into a grid of square bins and performs layer generation within each bin. Choosing the optimal bin size significantly increases performance because objects that are distant from one another are not involved together in layer-generation operations. |
| **Bin size** | Length, in display units, of each side of a bin. |
| **Merge objects after generation** | Causes objects on a generated layer to be merged upon completion of the process. This option can significantly increase processing time for more complex layouts. |

**Warning:** If a source layer (i.e., an input to a derived layer) is hidden, L-Edit ignores objects on that layer.

When you execute the **Generate Layers** command, L-Edit automatically deletes existing objects on derived layers before regenerating those layers. If you generate only a single layer, however, L-Edit does not delete objects on other derived layers.

To delete such objects, check the option **Delete all derived layers prior to generation**. L-Edit does not delete ports on derived layers. If generation is disabled for a particular layer, L-Edit does not automatically delete objects on that layer before generating other layers.

**Working with 45° Objects**

When layers are generated with **Tools > Extract**, all off-grid vertices are rounded to the nearest internal unit that preserves the angles. Off-grid vertices may result from off-grid intersections of 45° polygons, and from conversion of 45° wires to polygons. The coordinates of off-grid vertices are rounded to the nearest internal unit while still preserving 45° and 90° angles. If the dimensions of the source objects (measured in internal units) are small, then the resulting polygons may be distorted.

In the following example, the polygons on layers a and b create an off-grid intersection. (The distance between gridpoints is one internal unit). When generating a polygon equal to (a | b), L-Edit rounds the
off-grid vertex to the nearest gridpoint. To preserve 45° and 90° angles, two additional vertices must be shifted to the left. This results in a distortion of the original shapes.

To prevent distortion from rounding of grid coordinates, you should maintain a minimum resolution, in internal units, for all edge lengths, wire widths, distances between objects, and Grow distances. You can specify this minimum resolution by setting the mapping from internal units to technology units in the Setup > Design—Technology dialog. A setting of at least 100 internal units per technology unit is recommended for designs containing 45° objects.

When you run Extract, L-Edit checks the snap grid parameter. If it is less than 100 internal units, L-Edit assumes that edges or spacings smaller than 100 internal units may exist on the layout, and the warning appears, suggesting that the layout may need to be rescaled.

If you are certain that no edges, wire widths, spacing, or Grow values smaller than 100 internal units exist, then click Yes to proceed. Changing the snap parameter to an equivalent of 100 internal units prevents the warning from appearing (see “Setup Design–Grid” on page 82).

The equivalent number of internal units is based on the snap grid value (in display units) and the number of internal units per display unit. It is up to you to make sure that no objects smaller than the subgrid are created; the warning depends only on the current value of the snap grid, not on the actual size of objects in the layout. You can rescale the design by increasing the number of internal units per technology unit.

**Writing Out Node Names**

Legacy Extract can write out nodes as internally generated numbers (using the option Integers in Extract—Output) or as descriptive strings (using the option Names in Extract—Output). For further information on this dialog, see “Setup Extract Standard Rule Set–Output” (page 1984).

To label a node or element for extract to a netlist, you must add a port to the layer of that node or element, within an object (box, polygon or wire) on that layer. When Names is selected, L-Edit derives node names from the names of ports found on the same layer as the node. It derives element names from
ports found on the device-specific recognition layer that are completely enclosed by that device. Port labels are transferred when generating a layer for extract. For example, if you have a node name IN1 that is on Metal 1 but in your extract definition file you use M1Wire which is Metal 1 and Not Inductor ID, then the port on Metal 1 will be transferred to M1Wire during extract and will label the node IN1.

If you want to use the same ports for a design rule check that uses an assigned dummy layer, you can change the layers for your ports by:

☑ hiding all objects except for ports
☑ hiding all layers except the node layer
☑ selecting all objects, which will be just the ports on the node layer
☑ using Edit > Edit Object to change the layer to the layer of your choice.

The strings produced by the extractor are the hierarchical names; each instance involved in a node is mentioned and separated from the others by a slash (/), with the port name at the end. (Instances that are unnamed in the layout are named automatically by the extractor.) For example, the node name U1/alpha/in describes a port in contained by an instance alpha, which in turn is contained by an instance U1.

**Adding User Parameters to Extracted Devices**

Users can annotate extracted devices with parameters that are meaningful to downstream tools. To annotate a device, create a port on the recognition layer that overlaps the device in question. The port should be the device name followed by user parameters that users want add. This will set the particular name and user parameters for that device. The first word of the port will become the SPICE name for the device and the rest of the port text as the parameters to add to the device line.

Users can add extra parameters, replace existing parameters, or use existing parameters in their expressions of their extra parameters. The directory L-Edit Pro\Samples\Extract contains two examples demonstrating user parameters. The following are a series of examples showing the different results of user parameters.

- **No port label**
  R23 nodeA nodeB R=3

- **Port text = "Rin" - labels a device to be Rin**
  Rin nodeA nodeB R=3

- **Port text = "Rin turns=4" - labels a device to be Rin, with property "turns=4"**
  Rin nodeA nodeB R=3 turns=4

- **Port text = "turns=4" - adds property "turns=4" to auto-named device**
  R23 nodeA nodeB R=3 turns=4

- **Port text = "Rin R=10 turns=4" - labels a device to be Rin, with property "turns=4" and the "R=" property will be replaced with the value in the port text.**
  R23 nodeA nodeB R=10 turns=4

- **Port text = "Mn1" - labels a device to be Mn1 - The model name is taken from the extract definition file.**

  ```
  device = MOSFET{
    RAYER=ntran;
    Drain=ndiff, AREA, PERIMETER;
    Gate=poly wire;
    Source=ndiff, AREA, PERIMETER;
  }
  ```
Bulk=subs;
MODEL=NMOS;
)

Mn1 nodeD nodeG nodeS nodeB NMOS L=5u W=10u

- Port text = "Mn1 MODEL=MyNMOS" - labels a device to be Mn1, with the model name replaced with the specified model name, "MyNMOS". This allows the user to override the model name in the EXT file.

Mn1 nodeD nodeG nodeS nodeB MyNMOS L=5u W=10u

- Port text = "Mn1 AD='${W}*6u'" - labels a device to be Mn1, with ${W} replaced with the width of the transistor. This allows the user to use device parameters in expressions of other parameters.

Mn1 nodeD nodeG nodeS nodeB MyNMOS L=5u W=10u AD='5u*6u'

The port string can include tokens, which are references to other values and device parameters. The following tokens are expanded when writing to the SPICE netlist. All other text is parsed without expansion. See the previous paragraph for an example.

<table>
<thead>
<tr>
<th>Token</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>${property}</td>
<td>The value of the named property (as a string). property can be expressed using the full path—e.g., Mechanical.Length. If no path is designated, EXTRACT is assumed.</td>
</tr>
<tr>
<td>\n \t</td>
<td>New line (\n) or tab (\t) characters.</td>
</tr>
<tr>
<td>\ $ {</td>
<td>}</td>
</tr>
</tbody>
</table>
Legacy Extract

**Setup Extract Standard Rule Set—General**

Use Tools > Add-Ins > Legacy Extract... to access this tab so you can specify input and output files and other primary settings.

- **Extract definition file**
  - Name of the input file containing the extractor device and interconnection definitions. You can choose from available files and directories with the Browse button. You can open the file in a text window with the Edit button. This will also close the Extract dialog.

- **SPICE extract output file**
  - Name of the output file containing the extracted netlist. Enter the name (or use the default). You can choose from available files and directories with the Browse button. You can open the file in a text window with the Edit button. This will also close the Extract dialog.

- **Overwrite existing output files**
  - When checked, causes automatic overwriting of the SPICE output file, even if these files already exist.

- **Open SPICE output file after extracting**
  - When checked, automatically opens the SPICE output file after Extract runs.

- **Label all devices**
  - For each unnamed device, creates a two-dimensional port at the location of the device. The text of the port is the text of the element name for the device. Device labels will not be generated for devices with user-placed labels.
  - The group Place device labels on layer contains options for writing the device labels on the device-specific Recognition Layer or another layer you select.

- **Nodal parasitic capacitance**
  - Ignores nodal parasitic capacitance tolerances.
  - Enter nodal parasitic capacitance tolerances.
**Write nodal parasitic capacitance**

Computes the capacitance with respect to the substrate of each node in the circuit using the area and fringe capacitance constants specified with “General Layer Parameters” on page 91. The node to substrate capacitance of \( N \) is written to the netlist as a capacitor between \( N \) and the substrate/ground (0). The form of this notation is \( C_{par1}, C_{par2}, \text{etc.} \).

**Write nodal parasitic capacitance** is not generally turned on when a netlist is extracted for LVS since the other netlist (typically derived from a schematic) will not contain parasitic capacitors associated with nodes.

**Ignore nodal parasitic capacitance less than**

Specifies a limit, in femtofarads, below which the nodal parasitic capacitance will not be written to the netlist. This field is disabled when the **Write nodal parasitic capacitance** box is unchecked.

**Bin size**

Length of one side of a bin, in display units. To improve performance, L-Edit divides the layout into a grid of square bins and extracts each bin individually. Devices that cross bin boundaries are extracted properly.

Binning is not used if the **Recognize subcircuit instances** box in the **Subcircuit** tab is checked.
Setup Extract Standard Rule Set–Output

Use the fields in this tab to specify the way in which the extracted circuit is written to the output netlist.

Write node names aliases

Writes all node names associated with each node in comments at the beginning of the netlist file in the section **NODE NAME ALIASES**.

* NODE NAME ALIASES
* NODE = NODE NAME (NodeLabelX, NodeLabelY)
* 1 = U1/Out (39.5,-9)
* 2 = B (-1,23.5)
* 2 = C (-42.5,23.5)
* 2 = U0/A (13.5,-9)
* 3 = D (33.5,20)

Write device coordinates

(Display units)

Writes the coordinates in display units of the lower left and upper right corners of the device at the end of the SPICE line as a in-line comment.

M1 1 3 5 5 PMOS L=2u W=28u $ (36.5 29 38.5 57)

P-Spice compatible comments

Writes in-line comments using the ; character instead of the $ character so that they are compatible with P-Spice.

M1 1 3 5 5 PMOS L=2u W=28u ; (36.5 29 38.5 57)

Write terminal names for subcircuits

Writes the terminal names for subcircuits in a comment following each subcircuit statement in the netlist. Terminal names of other devices are never written to the netlist file.

X1 1 2 4 IResPoly L=2.4u W=720n
* X1 PLUS MINUS BULK

Write shorted devices

If IGNORE_SHORTS is set in the extract definition file, writes shorted devices into the netlist as comments; otherwise, shorted devices are ignored. If IGNORE_SHORTS is not set, a shorted device is written to the SPICE file as a regular device.
Write layer cap. & resistance warnings

Writes warnings on missing layer capacitance and resistance values to the specified netlist file.
* Warning: Layers with Unassigned AREA Capacitance.
  * <PMOS Capacitor ID>
* Warning: Layers with Unassigned FRINGE Capacitance
  * <Poly1-Poly2 Capacitor ID>
* Warning: Layers with Zero Resistance.
  * <N Well Resistor ID>

Write node-element cross-reference

Writes a node-element cross-reference table in the comments of the netlist file. Also, list are the element’s terminal, the edge between the node and the device recognition polygon, and the device’s location.

* NODE = ELEMENT Terminal (PinEdgeX1 PinEdgeY1 PinEdgeX2 PinEdgeY2)
  * 6 = R1 P (-65 20 -65 30) (-65 20 -60 30)
  * U0/A = R1 N (-60 20 -60 30) (-65 20 -60 30)
  * U0/A = Mn1 G (18.5 -9 20.5 19) (18.5 -9 20.5 19)
  * U0/A = Mp2 G (18.5 29 20.5 57) (18.5 29 20.5 57)
  * U0/Gnd = M2 B (36.5 -9 38.5 19) (36.5 -9 38.5 19)
  * U0/Gnd = Mn1 B (18.5 -9 20.5 19) (18.5 -9 20.5 19)
  * U0/Out = M1 D (36.5 29 36.5 57) (36.5 29 38.5 57)

The element terminal abbreviations are as follows:
- Resistor, Capacitor, Inductor: P - Positive, N - Negative
- Diode: P - Anode, N - Cathode
- BJT: C - collector, B - Base, E - Emitter, S - Substrate.
- JFETs/MESFETs: D - Drain, G - Gate, S - Source, B - Bulk
- MOSFETs: D - Drain, G - Gate, S - Source, B - Bulk

For Subcircuit, the entire pin name from the EXT file is written.

* 1 = X1(MINUS) (-16 25 -16 36) (-18.5 25 -16 36)

Write node name conflicts not at the top level

Flags node name conflicts that occur in the hierarchy other than in the top level. This is especially helpful for nodes in standard place and route cells.

Write nodes and devices as

Controls whether nodes are written as internally generated numbers (Integers) or as descriptive strings (Names). Ports in the layout can be used as node or element names in the netlist. For further information, see “Writing Out Node Names” on page 1979.

Write values in scientific notation

When checked, this option writes numerical values in scientific notation instead of SPICE engineering units.

Option on (checked)

<table>
<thead>
<tr>
<th>Device</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>1 3 5 5 PMOS L=2E-6 W=2.8E-5</td>
</tr>
</tbody>
</table>

Option off (unchecked)

<table>
<thead>
<tr>
<th>Device</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>1 3 5 5 PMOS L=2u W=28u</td>
</tr>
</tbody>
</table>

Write verbose SPICE statements

Writes resistors, inductors, and capacitors to the netlist file with the device value preceded with a R=, L=, or C=. For example, a capacitor would have the following format: Cxxx n1 n2 modelName C=cValue.

Write empty subcircuit definition

Writes an empty subcircuit definition block at the top of the netlist file. Use only with Recognize subcircuit instances on “Setup Extract Standard Rule Set–Subcircuit” (page 1986).

Write .END statement

Writes a .END statement at the end of the netlist
**Wrap lines greater than 80 chars.**

Wraps SPICE lines at 80 characters and continues them on the next line, including comments.

M189 DataInB DataInUnBuf Vdd Vdd PMOS L=2u W=13u becomes
M189 DataInB DataInUnBuf Vdd Vdd PMOS
+ L=2u W=13u

**SPICE include statement**

Specifies text that is written unaltered as the second line of the output netlist. Typically, an `.include file` command is entered, where `file` represents a model or subcircuit file name.

**Note:**

L-Edit cannot determine if other nodes in the circuit are ground nodes. If other nodes are to represent ground, then they must be renamed 0—or any of its equivalents—in the netlist.

**Setup Extract Standard Rule Set—Subcircuit**

Use the fields in this tab to specify parameters for subcircuit extraction.

**Recognize subcircuit instances**

Activates the subcircuit recognition feature.

**Subcircuit recognition layer**

Name of the subcircuit recognition layer (SRL). This mandatory layer should not contain electrically significant geometry.
### Write netlist as a subcircuit definition (.SUBCKT ... .ENDS)

When checked, this option writes the entire netlist in subcircuit format. A `.subckt` command appears before the first device, and an `.ends` command appears after the last device. When this option is used, there must be a subcircuit recognition polygon at the top level defining the subcircuit in order for Extract to proceed with subcircuit extraction.

### Flag improper overlaps

Controls the reaction to geometry violations: under- or over-filled connection ports or geometry that overlaps the subcircuit boundary. Check to display warnings; clear to suppress warnings. Suppressing warnings can be useful when extracting autorouted standard cell designs with known over- and under-fill characteristics.

### Ignore subcircuit connection ports with names

List of ports whose names are ignored in subcircuit extraction. Contains the following fields:

- **SPR core ports** — read-only field listing port names predefined in SPR Core Setup—General.
- **SPR padframe ports** — read-only field listing port names predefined in SPR Padframe Setup—General.
- **One Other Port** — use this field to specify one additional subcircuit connection port to be ignored.
- **Ignore subcircuit connection port on layer** — name of a layer on which intruding geometry and subcircuit connection ports will not be recognized. The netlist extractor also ignores any geometry on the Icon layer (often used for documentation purposes).

For more information see “Designing Subcircuit Cells” on page 1988.

### Subcircuit cross port names

Lists ports whose names are ignored in subcircuit extraction. Contains the following fields:

- **SPR row crosser** — read-only field listing subcircuit cross ports predefined in SPR Core Setup—General.
- **One Other Port** — use this field to specify one additional subcircuit cross port.

For more information see “Crossing Over a Subcircuit Instance” on page 1992.

### Activating Subcircuit Recognition

Subcircuit recognition is activated by checking the **Recognize subcircuit instances** option in the dialog “Setup Extract Standard Rule Set–Subcircuit” (page 1986).

If the **Write netlist as a subcircuit definition** option is checked, then the entire netlist is written in subcircuit format:

- A `.subckt` command appears before the first device statement, and an `.ends` command appears after the last device statement.
- Subcircuit connection ports at the top level (that is, not contained in instances) of the extracted cell are written as SPICE subcircuit pins in the output.
This feature can provide complete subcircuit definitions corresponding to subcircuit instance statements generated from other cells. It requires that the subcircuit recognition polygon and the proper pin ports exist at the top level.

As the extractor runs with subcircuit recognition activated, any errors are reported, and ports placed on the Error layer at their locations in the layout.

**Using Subcircuit Definitions for Hierarchical Cell Recognition**

Though Legacy Extract does not process hierarchical designs, it does allow for a form of hierarchical extraction that speeds the extraction process in higher-level cells. This is done by designating often-instanced lower-level cells “subcircuit cells,” essentially making them black boxes, so that every instance will not be extracted explicitly.

When *not* set to recognize such “subcircuits,” Legacy Extract flattens instances. The extracted netlist therefore describes all devices at the same level, with no indication of hierarchy. However, if subcircuit recognition *is* activated and there are instances of subcircuit cells, then the extracted netlist contains:

- An empty subcircuit definition block corresponding to each subcircuit cell. Each such block begins with the `.subckt` command and ends with the `.ends` command. Subcircuit and node names in the netlist are taken from the names of the subcircuit cells and their connection ports.

- A SPICE subcircuit instance statement corresponding to each instance. Each such statement has the form `xinstance pin1 ... subcircuit`, where `instance` represents the instance name, `pin1` ... the pin list, and `subcircuit` the subcircuit definition name. (If the instance is unnamed in the layout, Extract automatically assigns its name in the netlist.)

**Designing Subcircuit Cells**

Subcircuit recognition is recursive within non-subcircuit instances. If a higher-level cell contains a non-subcircuit instance, and the instanced cell itself contains marked (subcircuit) instances, then the subcircuit instances are properly extracted as subcircuits at any level of hierarchy, and any non-subcircuit instances are flattened.

**Subcircuit Recognition Polygons**

A cell is marked as a subcircuit cell by the presence of a subcircuit recognition polygon (SRP) on the subcircuit recognition layer (SRL).

The SRP is a box or a 90° polygon. It delimits the area of any of the subcircuit cell’s instances that cannot be overlapped by geometry in the containing cell and the perimeter at which subcircuit connection ports may be placed. There are two exceptions to the rule against overlapping an instanced cell’s SRP:

- Geometry inside subcircuit connection ports.
- Geometry over cross port channels.

There may be only one SRP per subcircuit cell. If no SRP exists in the cell, then its instances are not recognized as subcircuits; the extractor flattens them.

Any geometry in a cell that contains an SRP, including geometry outside the SRP, appears in instances of the cell but is ignored by the extractor.
Subcircuit Connection Ports

The pins of a subcircuit instance are formed by placing *subcircuit connection ports* inside the subcircuit cell on the particular layer on which connections will be made to the instance. A connection port must both (1) be completely contained by the SRP, and (2) share an edge with the SRP. The port may be 2-dimensional or 1-dimensional (as long as it is co-linear with an SRP edge), but not 0-dimensional (a point).

The text associated with a connection port is transferred to the output netlist as the name of a signal parameter (node) on the subcircuit definition. All connection ports, on all layers, with the same name (within one subcircuit cell) are extracted as the *same* subcircuit pin. The pins of a subcircuit are written in alphabetical, then numerical, order.

Certain named ports can be ignored as candidates for connection ports. These are shown in the dialog “Setup Extract Standard Rule Set–Subcircuit” (page 1986), in the *Ignore subcircuit connection ports with names* section:

<table>
<thead>
<tr>
<th>Ignored ports</th>
<th>How specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPR core ports</td>
<td>“SPR Core Setup–General” (page 1927)</td>
</tr>
<tr>
<td>SPR padframe ports</td>
<td>“SPR Padframe Setup–General” (page 1937)</td>
</tr>
<tr>
<td>Ports on the Icon layer</td>
<td>“Rescaling a Design” (page 130)</td>
</tr>
<tr>
<td>Ports matching a single additional name</td>
<td>Other text field</td>
</tr>
<tr>
<td>Ports on a single additional layer</td>
<td>Ignore subcircuit connection port on layer drop-down list</td>
</tr>
</tbody>
</table>

Connecting to a Subcircuit Instance

A connection to a subcircuit instance is formed by drawing an orthogonal wire, box, or polygon into a connection port, on the same layer.

- **Into a 1-dimensional** port, connecting geometry should be *exactly* as wide as the port and must *exactly* abut the port without overshooting it.
- **Into a 2-dimensional** port, connecting geometry should *exactly* fill the port, with neither gaps nor spillovers.

Odd-width wires with extend or round end styles should not be used.
If the connecting geometry touches the connection port but does not exactly satisfy the above criteria, then a connection is still specified in the output netlist, but a warning is generated.

The SRPs of multiple subcircuit instances may be abutted together; connections are formed between abutting 1-dimensional connection ports without additional geometry.

Connecting geometry should approach the SRP orthogonally for a distance at least equal to the largest DRC spacing rule specified for the connecting layer. Non-connecting geometry should not be placed any closer to a subcircuit instance than this distance.
A connection port should not exist on an “inside” corner of an SRP, but should be separated from the corner by a distance $D$ at least equal to the largest spacing rule value $L$ specified for the layer. See the following illustration.

**Note:** Extract does not check spacing rules.

Subcircuit connection ports and SPR signal connection ports have very similar functions: they mark the locations of connections from outside to inside instances. There are, however, some important differences.

- Subcircuit connection ports may be 1- or 2-dimensional. SPR signal ports must be 1-dimensional.
- 2-dimensional subcircuit ports must be completely filled by connecting geometry, and 1-dimensional subcircuit ports require the connecting geometry to have the same width. SPR signal ports can be of a different width than the connecting geometry.
- SPR signal ports may be entirely within the interior of a cell. Subcircuit connection ports must share an edge with the subcircuit recognition polygon.

Because of these factors, the port construction shown in the following figure is used in standard cell design for use with both SPR and Extract. Moreover, running Extract with the **Flag improper overlaps**
option turned off eliminates subcircuit extract warnings. For further information, see “Setup Extract Standard Rule Set–Subcircuit” (page 1986).

Crossing Over a Subcircuit Instance

A 1-dimensional cross port in a subcircuit cell defines a “channel” in the cell’s instances, over which geometry may run without causing an overlap warning. The channel runs perpendicular to the width of the cross port and extends from one end of the subcircuit recognition polygon to the other.
Extract recognizes the following as cross ports:

<table>
<thead>
<tr>
<th>Cross ports</th>
<th>How specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row crosser ports</td>
<td>“SPR Core Setup—General” (page 1927)</td>
</tr>
<tr>
<td>Ports matching a single additional name</td>
<td>In “Setup Extract Standard Rule Set—Subcircuit” (page 1986), in the Subcircuit cross port names section, Other text field</td>
</tr>
</tbody>
</table>

SPICE OUTPUT Properties

SPICE OUTPUT properties are a subset of the general purpose L-Edit properties described in “Properties” on page 123. In subcircuit extraction, SPICE OUTPUT properties determine the device name, connectivity, and device parameters of the subcircuit.

You can use properties to format the output of subcircuit information written to a SPICE netlist. You can attach properties to either a parent cell or an instance. L-Edit Extract searches for instance properties first. If it does not find any, it searches for properties on the parent cell.

SPICE OUTPUT properties are only processed as part of subcircuit extraction. See “Using Subcircuit Definitions for Hierarchical Cell Recognition” on page 1988 for more information.

Property Tokens

String properties can include tokens, which are references to other values and variables. The following tokens are expanded during subcircuit extraction. All other text is parsed without expansion.

<table>
<thead>
<tr>
<th>Token</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>An incremented integer that counts the instances of the cell. (This token is only expanded during extract.)</td>
</tr>
<tr>
<td>${property}</td>
<td>The value of the named property (as a string). property can be expressed using the full path—e.g., Mechanical.Length. If no path is designated, EXTRACT is assumed.</td>
</tr>
<tr>
<td>%port</td>
<td>The name of the node to which the pin associated with the named port is attached.</td>
</tr>
<tr>
<td>\n \t</td>
<td>New line (\n) or tab (\t) characters.</td>
</tr>
<tr>
<td>\ # $ % { } [ ]</td>
<td>The character after the initial backslash (instead of being interpreted as part of a token).</td>
</tr>
</tbody>
</table>

For example, the value of the following EXTRACT.SPICE OUTPUT property:

XPlate %{right} %{left} platemodel W=${W} L=${L}

would result in the following netlist output:

XPlate 5 3 platemodel W=5e-6 L=2e-6

where EXTRACT.W=5e-6 and EXTRACT.L=2e-6.
**Application Example**

In the following example, the **SPICE OUTPUT** property allows you to specify multiple energy domain connections such as electrical and mechanical connections—for example, the **SPICE OUTPUT** property of the following MEMS plate:

\[
X${\text{${\text{instance}}} %${PL\_Left}_m %${PL\_Right}_m %${PL\_Bottom}_m %${PL\_Top}_m
   %${PL\_Left}_e %${PL\_Right}_e %${PL\_Bottom}_e %${PL\_Top}_e mass4\_geo W=${W} L=${L}
\]

will result in the following netlist output:

\[
Xu5 3_m 4_m 6_m 2_m
   3_e 4_e 6_m 2_m
   mass4\_geo W=3e-3 L=2e-4
\]

where **EXTRACT.W=3e-3** and **EXTRACT.L=2e-4**.

Notice that all ports are referenced twice in this string using property tokens: one to specify the mechanical connection and the other to specify the electrical connection.

**Cells to flatten**

Use this list to select the cells that will be analyzed for flattening prior to DRC, extract and node highlighting. Use the **Mark All** and **Unmark All** buttons to speed the selection process.
Index

SYMBOLS

.include statement, 984
.model statement, 985

NUMERICS

3-D data
  in DXF files, 139
45-degree layout
  design rule checking, 550
45-degree objects in generated layers, 530, 1978

A

abutment ports
  in pad cells, 1964
  in standard cells, 1960
abutment ports (SPR)
  in core cells, 1919
abutment type, 1813
activating drawing tools, 279
active net, 494
acute angles, 539
add mode, 96
adding a macro, 1069
adding wire sections, 299
Add-Ins
  GDS Properties, 159
  I/O Pad Cross Reference, 267
  Legacy Layer Generation, 529, 1977
  PostScript Mask Separation, 153
  Text Resize, 332
  Transfer File Info to Cells, 123
algorithms, 1027
Alignment toolbar, 177
all-angle
drawing tools for, 280
ancestor cells, 218
AND operation, 447
antenna checks, 788
area
  with guard rings, 403
arrays, 256
  aligning to manufacturing grid, 305
  by aligning objects, 343
creating, 256, 316, 328
creating from regular cells, 66
creating in Edit > Objects, 256
editing, 252
editing by stretching, 66
GDSII name for, 160
generated, 400
grouping instances to create, 316
in T-Cells, 376
instances, comprising, 256
instances, creating from, 256
of vias, 400
ungrouping, 317
using the command line, 367
visibility of, 189
arrays, creating, 328
ASIC design, 1017
assisted routing, 391, 398
  adding existing via cells, 397
attached libraries, 49
auto-generated cells See T-Cell
automatic layer generation, 460
automorph classes, 971, 1024
detailed trial matching, 1025
example of detailed trial matching, 1012
fanout, 1012
hierarchical information, 1012
output example, 1011
parameter matching, 1026
prematch files, 1025
resolving fragmentation, 1013
auto-panning, 207
auto-routing
  breakouts, 488
  non-routing layers, 462
  routing layers, 462

B

base point
  locating by coordinates, 325
  moving objects with respect to, 322
  setting, 324
Base Point toolbar, 175, 176, 177, 182, 183
batch-file syntax, 1030
batch files, 1004
batch mode, 1004
bevel style (wire joins), 297
bin size
  with Extract, 1983
  with Generate Layers, 919, 1978
binding macros
  to hot keys, 1049
  to menu items, 1049
BJT
  syntax in EXT file, 929
Boolean operators for generated layers
  AND, 447
  Grow, 448
  NOT, 447, 453
  OR, 447, 452
  order of operations, 449
  shrink, 448
Boolean/Grow Operations, 345
boxes
  drawing, 281
  editing, 354
  merging, 345
  nibbling, 344
breakouts
  placed by the autorouter, 484
buffer cells, 1922, 1967, 1968
  input ports in, 1968
butt style (wire ends), 297
buttons
  adding to toolbars, 180
  creating customized for toolbar, 180
  renaming, 180
  TCL functions for, 180

C

Calibre results in L-Edit, 954
Calibre Results Viewing Environment (RVE), 959
callbacks
  for T-Cells, 383
calls
  in CIF, 154
CAP format, 1975
capacitance, 1953
  calculation (SPR), 1953
capacitor
  syntax in EXT file, 927
CDL Files, 1004
Cell, 216
Cell >
  Close As, 233
  Copy, 237
  Delete, 239
  Fabricate, 267, 273
  Flatten, 257, 258
  Info, 216
  Instance, 250, 251
  New, 227
  Open, 229
  Rename, 232
  Revert Cell, 230
  Save Cell to TDB File, 233
cell hierarchy
  bottom up, 245
  display, 243
  display, copy to text view, 244
  printing, 244
  top down, 245
  viewing, 242
cell mapping (SPR), 1918, 1924
cell-cell spacing (SPR), 1931
cell-power spacing (SPR), 1931
cells, 218
  ancestors, 218
  arrays, 256
  buffer, 1922
  children, 218
  chip cell in SPR, 1943
  clustering, 1933
  copying, 236, 249, 328
  copying a region of, 234
  copying a section of, 234
  copying information from files, 123
  creating, 227
  deleting, 239
  descendants, 218
  excluding from DRC, 554
  fabrication, 247
  flattening, 317
  generated, See T-Cell
  ground pad, 1937, 1965
  information, 216
  instancing, 249, 251
  leaves, 218
  listing layers with geometry, 242
  listing objects in by layer, 241
  locking, 44, 45, 247, 265
  new, 227
  opening, 229
  parents, 218
  renaming, 231
  replacing, 264, 265
  reserving, 44
  reverting, 230
  saving, 230
  showing and hiding, 222, 229, 244, 252
  standard, 1914
  tie-to-ground, 1927
  tie-to-power, 1927
  version number, increasing & decreasing, 216
  view type, 215
views, 215  
chamfers  
  adding, 286, 332  
  setting the distance value, 287  
channel base name, 1928  
channel pitch, 1930  
child cells, 218  
chip cells in SPR, 1943  
CIF, 153  
  calls, 154  
  extensions, 148, 157  
  forward references, 157  
  geometric primitives, 154  
  layer names, 156  
  layers, 156  
  restrictions, 157  
  scaling, 158  
  symbols, 153  
  wires, 157  
CIF files, 133, 136, 146, 148, 151, 152, 153  
  Exporting CIF Files, 148  
  Importing CIF files, 136  
CIF format  
  compatible wire styles, 298, 373  
  fabrication cell, 266  
circles  
  approximating in GDSII, 161  
  approximation in GDSII, 147, 158  
  drawing, 281  
  drawing mode to use, 280  
  editing, 357  
  in CIF, 148  
  snapping to the manufacturing grid, 305  
class separations, 1023  
clear mode, 96  
clearing objects on generated layers, 461  
clipout region for cells, 233  
clock routing, 1921  
clusters (SPR)  
  illustrated, 1933  
color index, 60  
color parameters, 60  
column chooser, 220  
command line commands  
  !!, 367  
  array, 367  
  box, 368  
  copy, 369  
  goto, 369  
  instance, 369  
  layer, 370  
  move, 370  
  paste, 371  
  path, 370  
  polygon, 371  
  run, 372  
  sc, 367  
  text, 372  
  width, 373  
command line scripting, 373  
command-line arguments, 47  
  adding to toolbars, 180  
  creating customized, 180  
  renaming, 180  
  TCL functions for, 180  
commuter licenses, 166  
commuter licensing, 165  
comparing fragmented classes, 1024  
compiled macros  
  creating, 1045  
  composition, 1811  
  composition direction, 1813  
configuration files  
  contents, 62  
  editing, 62  
  saving, 62  
connecting segments  
  DXF import, 140  
consider parameters (LVS command-line option), 1031  
constraints  
  in SPR core generation, 1949  
  contact cells  
    generated for routing, 391  
Converting Objects to Polygons, 347  
converting polygons to boxes, 136  
coordinate system  
  display, 66, 185  
  units, 186  
coordinates, 186  
copying  
  cells, 236, 249  
  images, 121  
  objects, 328  
  to the Windows clipboard, 328  
copying layout, 121  
copyrights  
  adding to layout, 127, 128, 129  
core generation  
  constraints in SPR, 1949  
  design rules, illustrated, 1931  
  core with signal ports, illustrated, 1919  
corner pad cells, 1966  
Create/Update T-Cell Code, 248
creating
  compiled macros, 1045
  interpreted macros, 1046
layout palette, 1060
resources for layout palette, 1061
critical nets, 1971
  in SPR, 1932, 1973
criticality, 1932
  in EDIF, 1973
cross ports, 1992
cross-section
  with etch, 209
  with grow/deposit, 209
  with implant/diffuse, 210
operation of, 210
process definition files for, 209
process steps, 209
single-step display, 212
curved polygons
  over 200 vertices, 145
straightening curves, 280
curves
  adding fillets, 286, 332
  approximating for manufacture, 305
  changing radii, 332
  changing sweep angle, 332
  converting for GDSII, 145
  converting polygons to, 284
  converting to polygons in DXF, 151
  converting to straight-sided polygons, 145
curve height, 285
CurveTools, 286
displaying the curve height, 355
drawing, 284
drawing mode to use, 280
in CIF, 148
in GDSII, 145, 158
snapping to the manufacturing grid, 305
straightening curved segments, 350
CurveTools, 286
customer support, 165
customized elements, 1020

D
data types
  GDSII, 160
data.dm files, 47
debugging macros
  interpreted, 1049
decendant cells, 218
default text size, 87
deleting
  cells, 239
  objects, 314
  objects on generated layers, 461
density design rules, 538
deselection
  cycling, 312
  edge, 312
  explicit, 311, 313
  hidden layers, 314
  implicit, 312, 314
  range, 85
  universal, 314
design file in SPR, 1916
design flow
  in SPR, 1916
Design Navigator, 242
  Show all cells, 242
design rule checking
  45-degree layout, 550
  clearing error markers, 949
density rules, 538
  exact width rules, 536
  Excluding Cells from DRC, 554
  extension rules, 537
  finding error markers, 949
generated layers, 534
  minimum width rules, 535
  not exist rules, 536
  optimization, 545
  overlap rules, 537
  performance optimization, 545
  region-only check, 550
  rule exceptions, 538
  rule sets, 547
  setup, 533
  spacing rules, 536
  surround rules, 536
types of rules, 535
design rule sets
  creating, 533
ingediting, 533
merging, 533
design rule types, 535
design rule violations
  clearing, 949
  finding, 949
design rules
  core (SPR), 1930
density, 538
  exact width, 536
  exceptions, 538
  extension, 537
  minimum width, 535
  not exist, 536
  overlap, 537
  pad routing, 1944
  spacing, 536
  specifying, 542
surround, 536
designs
  flat, 249
detailed trial matching, 1025
    example, 1012
    output, 1013
Details button, 185
diagnostic functions
  running, 165
diode
  syntax in EXT file, 930
display
  updates to, 71, 1070, 1124–1125
display grid, 82
displaying layout interface elements, 188
Draw
  Clip Out Region, 233
  Convert-Change Layers, 327
  Layout Text Generator, 126, 127, 128
  Wire Utilities, 299
Draw >
  Add Wire Section, 299
  Flip, 326
  Force Move, 324
  Group, 316
  Layout Generator, Import Images, 143
  Merge, 345
  Move By, 322, 323
  Nibble, 344
  Nudge, 324
  Pick Layer, 279
  Rotate, 326
  Slice, 343
  Ungroup, 317
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
Draw>
  Base Point, 325
drawing
  all-angle objects, 280
  automatic selection, 85
  boxes, 281
  changing mode while drawing, 280
  circles, 281
  curves, 284
  directional constraints, 280
  layer constraints, 281
  objects, 280
  pie wedges, 281
  polygons, 281
  ports, 289
  rulers, 293
  tori, 281
  wires, 281
drawing objects, 215
Drawing toolbar, 173
drawing tools, 279
  selecting, 279
DRC
  disabling rules, 546
  Excluding Cells from DRC, 554
  false errors, 551
DRC rules
  Hole, 451
  inside, 451
  outside, 451
driving force, 1934
duplicating objects, 328
DXF files
  connecting segments, 140
  converting polylines, 140
  Exporting DXF Files, 150
  Importing DXF files, 139
DXF import
  connecting segments, 140
dynamic-link library, 1069

E

draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
draw operation
  nudging at fixed amount, 324
  selecting a reference point, 325
angle constraints, 280
boxes, 354
changing mode during operation, 280
circles, 357
directional constraints, 330
effective channel pitch, 1930
electrical models, 1975
element description file (LVS command-line option), 1032
element description file, example, 1020
elements, LComp, 1811
Elmore delay, 1953
engineering change orders (ECO), 463
errors
design rule checking, 545
exact width design rules, 536
example files
installing, 165
exchanging views, 207
explicit deselection, 313
explicit selection and deselection, 311
exporting files, 144, 146
CIF, 146, 148, 151, 152, 153
GDSII, 146
PostScript, 350
extend style (wire ends), 297
extending selection, 312
extending wires, 301
extension design rule, 537
extension design rules, 537
external verification interface (EVI), 959
Extract
General, 1982
Output, 1984
Subcircuit, 1986
extract definition file
comments, 925
connections, 925
devices, 926
extract definition files, 912, 924
extraction
capabilities defined, 912
configuring, 912
defining capacitance for resistors and capacitors, 920
defining devices and connections, 918
subcircuit recognition, 924, 1980, 1988

F
fabrication cell, 266
fabrication cells
I/O Pad Cross Reference, 267
fanout, 504, 988, 1012, 1934, 1935
fast iteration (LVS command-line option), 1034
File >
Close, 114
Exit, 132
Export Mask Data, 146, 148, 150, 152
Import Mask Data, 133, 136, 139, 142
Info, 122
Open, 108
Print, 115
Print Preview, 120
Print Setup, 119
Save, 110
Save As, 113, 114, 233
Setup > Import Technology, 54
file formats
CAP, 1953, 1975
CIF, 153
EDIF, 1970
GDSII, 158
LVS output file, 993
node and element list file, 994
SDF, 1938, 1974
SPICE, 994
TPR, 1969
XST, 213
file information
accessing, 122
transferring to cells, 123
file parameters, 88
files
exiting, 132
exporting, 144
importing, 133
locking, 973
opening, 107
saving, 109
fillets
adding, 286, 332
setting the distance value, 287
Find Next/Find Previous, 319
finding, 919
extract, 919
finding devices and nodes
node, 919
finding objects, 318
flattening hierarchy, 257
Flip/Rotate commands, 326
Force Move command, 324
forcing a move over an edit, 324
forward references
fracture
  on guard rings, 402
Fracturing Polygons, 350
fracturing polygons, 145, 147
fragmented classes, 1023
  comparing, 1024
  output, 1015, 1024
  resolving, 1016
fringe capacitance
  in capacitance file, 1953
  in SDF file, 1950

G

GaASFET
  syntax in EXT file, 930, 931
gaurd rings
  fracturing, 402
GDSII, 146
  arrays, naming, 160
  circle approximation in, 158, 161
  circle approximations in, 147
  curves, 158
  data type, assignment for ports, 361
  data type, assignment of, 162, 329
  data types, 92, 160, 161
  exporting GDS Files, 144
  file format, 158
  format, 160
    compatible wire styles, 298, 373
  format, wires, 162
  importing files, 133
  layer data type display, 198
  layer number display, 198
  layer numbers, listing, 99
  layer setup, 92
  numbers, 92
  port text sizing, 163
  properties, editing, 159
  wire formats, 162
  wire styles, 298, 373
GDSII files
  importing, 133, 136, 148
  generated cells, 374
    arrays, 400
    contact cells, 391
    guard rings, 400
    vias, 391
  generated layers
    45-degree objects and, 530, 1978
    AND operation, 447
    defining, 446
    design rule checking, 534
    for DRC and Extract, 460
    for extract, 529, 1977
    Grow operation, 448
    legacy, 529, 1977
    NOT operation, 447, 453
    OR operation, 447, 452
    removing generated layers, 461
    showing and hiding, 461
    shrink operation, 448
  generated cells for routing, 391
  geometric primitives
    in CIF, 154
  Gerber files, 142
    importing, 142
  global input signal routing, 1921, 1949
    cell design for, 1967
    illustrated, 1923
  global signal buses, 1921
  global signal ports, 1921, 1967
  global signal rail, 1922
  Goto, 207, 989
  granularity (LVS command-line option), 1033
  graphical repositioning, 321
  grid spacing
    minimum values, 483
  ground pad cell, 1937, 1963, 1965
  ground rail width, 1935
  ground signal
    in SPR, 1924
  grouping instances, 316
  grouping objects, 315
  Grow operation, 448
  guard rings
    area, 403
    generated, 400
  header files (for UPI), 71, 1070, 1071
Help >
  About L-Edit, 169
  hidden cells
    showing, 222, 229, 244, 252
  hidden layer
    indicator, 192
  hidden layers
    and geometry flags, 543
    hiding all, 192
    hiding all but one layer, 192
    hiding generated layers, 192
    hiding one layer, 192
    in Setup Layers dialog, 193
    indicator, 192
    ports on, 189
    selecting and deselecting, 314
    setting, 192
when copying cells, 233
hiding
  insides, 202
  layers, 192
  objects, 189
hierarchical design, 215
hierarchical information in automorph classes, 1012
hierarchical lock/unlock, 265
hot keys
  binding macros to, 1049
HStretch, 376

I

I/O Pad Cross Reference, 267
I/O signals
  in EDIF, 1972
  of SPR core, 1936, 1972
ignore bulk nodes (LVS command-line option), 1031
Images
  copying and saving, 121
images
  copying, 121
  saving, 121
implicit deselection, 314
implicit selection and deselection, 312
importing files, 133
  CIF, 133, 136, 148
  GDSII, 133, 136, 148
  Gerber, 142
  Laker format, 56
  Virtuoso format, 56
include files, 1071
indent middle rows, 1949, 1950
indent ratio, 1950
inductor
  syntax in EXT file, 929
ini file
  saving to a text file, 67
INI files
  contents, 62
  editing, 62
  saving, 62
initializing setup
  SPR, 1926
input file types, 972
inside, 451
insides
  hiding, 202
  showing, 202
instances, 215, 249
  arrays of, 256
  creating, 251
  editing, 252
  excluding from DRC, 554
  flattening, 257
  in hierarchical design, 215
  naming, 258
T-Cell parameters
toggling selectivity, 311
ungrouping, 317
instancing
cells, 249
interactive DRC
  reading and writing attributes, 50
internal units, 80
interpad connections, 1964
interpreted macros
  creating, 1046
  debugging, 1049
  running, 1071
intersection snapping, 306

J

JFET
  syntax in EXT file, 931
joining wires, 299
jump to schematic, 468

K

keyboard assignments
  writing to a configuration file, 67
keyboard shortcuts
  print to a text file, 67
keyword groups, 79
keywords, 79

L

label nodes with ports, 1950, 1952
launching L-Edit, 45
launching LVS, 971
layer
  legacy, 292
Layer fill, 529
layer palette
  filtering, 191
layer setup, 55
  importing from Laker, 56
  importing from Virtuoso, 56
layer spacing
  in core routing, 1931
  in pad routing, 1945
layer-cell spacing (SPR), 1931
layer-core spacing (SPR), 1931
layer-pad spacing (SPR), 1931
layer-purpose pairs
  reading and writing attributes, 49
layers
  adding, 104
  choosing active, 194
  CIF, 156
  CIF names, 156
  data type display, 198
  filtering display, 192
  GDSII layer number display, 198
  GDSII numbers for, 99
  generating, 529, 1977
  hidden, 198
  hidden, deselection of, 314
  hiding, 192
  layer name, 198
  layer number display, 198
  layer purpose number, 198
  listing, 99
  listing cells with geometry, 242
  listing types of objects on, 123
  locking, 198, 321, 461
  merging setups, 59
  moving objects from one to another, 327
  protected, 198
  protecting, 194
  rendering, 93
  reports on, 99
  selecting, 279
  setting icon size in palettes, 63
  showing, 192
  sorting in the palettes, 196
  special layers, 99
  valid, 91, 197
layer-to-layer capacitance, 1928
layout
  copying, 121
  saving, 121
layout area, 186
layout comparison, 904
layout files
  in SPR, 1917
Layout images
  copying and saving, 121
layout palette
  creating, 1060
  creating resources for, 1061
layout style (wire joins), 297
layout.oa file, 45
LComp functions, 1811
lcomp.h file, 1071
ldata.h file, 1071
leaf cells, 218
leaf-level, 202
L-Edit program icon, 45
ledit.ini file
  default location, 62
legacy layer, 292
Legacy Layer Generation, 1977
legacy ports, 291
leyers
  defining, 89
  lib.defs file, 44, 45
  format, 46
  lib.defs files, 47
libraries
  attached, 49
  locking, 44, 45
  referenced, 49
top-level, 45
licensing, commuter, 165
list elements and nodes (LVS command-line option), 1034
locator, 66, 185
  coordinate system, 82
  coordinates, 66, 185
  units, 66, 185
locked layers
  locking all, 193
  locking all but one layer, 193
  locking one layer, 193
log files (for UPI), 71
logo generation, 127, 128, 129, 554
logos
  adding to a design file, 1881
  copyrights
  adding to a design file, 126
  adding to layout, 127, 128, 130
LVS
  algorithms, 1027
  fragmented classes, 1024
  limitations, 1027
  model name syntax for, 985
  output, 1010
  output file format, 993
  probing, 1024
  removing parasitics, 984
  shorting nodes, 984
  verification errors, 1024
M
  Macro dialog, 1068
  macro dialog
    adding, 1069
  macros, 1043
    adding, 1069
  manual placement
    in SPR, 1933
Manufacturing Grid, 84
manufacturing grid
snapping circles to, 305
snapping curves to, 305
mapping cells and ports (SPR), 1918, 1924
mapping table (SPR), 1918, 1924
mask data
  importing to an active or new file, 133
mask polarity, 153
MAX_CELL_NAME, 1073
MAX_LAYER_NAME, 1073
MAX_TDBFILE_NAME, 1073
maximum geometrical difference (LVS command-line option), 1032
maximum value difference (LVS command-line option), 1032
MBB
  setting for an instance, 255
MEMS design example, 1020
menu items
  binding macros to, 1049
menus, 975
merging objects, 345
MESFET
  syntax in EXT file, 930, 931
minimum bounding box, 365
minimum width design rules, 535
mirror ports, 1967
mirroring, 1966
miter style (wire joins), 297
model names
  remapping, 1000
MOSFET
  soft connections, 923, 987
  syntax in EXT file, 932
MOSFETs
  permuted classes, 1017
mouse buttons bar, 185
Mouse snap grid, 84
mouse snap grid, 82, 177
Mouse Wheel Functions, 206
Move By, 322
move operation
  entering numerically, 322
  forcing, 324
moving
  objects, 320
  moving objects, 327
Multi Grid toolbar, 177
multiple object editing, 351

n
naming
  LComp elements, 1811
net criticality
  in EDIF, 1973
  in SPR, 1932
net delay
calculation (SPR), 1931
model formula (SPR), 1953
net length in SPR, 1975
net length reduction, 1950
netlist comparison, 1022
  resolving discrepancies, 1028
netlists
  EDIF format, 1918, 1970
  mapping (SPR), 1918, 1924
  TPR format, 1969
nibbling objects, 344
nodal capacitance, 1928
  files, 1953, 1975
nodal properties, 1953
node and element list file, 994
node highlighting
  mode enabled, 519, 520
  reading and writing attributes, 50
  saving data, 519, 521, 549, 916
node naming, 519
Nominal area, 929, 930, 931, 933
nonpolarized elements (LVS command-line option), 1036
not exist design rules, 536
NOT operation, 447, 453
number of rows, 1949
numeric repositioning, 322

O

Objects
  Boolean/Grow Operations, 345
c  converting to polygon, 347
  fracturing, 350
d  straighten curved segments, 280
objects
  aligning, 335
  boxes, 278
c  circles, 278
d  copying, 328
deleting, 314
deselecting, 313
drawing, 280
duplicating, 328
duplicating with an offset, 328
finding, 318
grouping and ungrouping, 315
hiding, 189
instances, 278
labels, 279
moving, 320
moving from layer to layer, 327
moving from one layer to another, 327
pasting, 328
pie wedges, 278
polygons, 278
ports, 278
reorienting, 326
resizing and reshaping, 329
rulers, 278
selecting, 311
showing, 189
slicing, 343
snapping, 304
text labels, 279
tiling, 335
tori, 278
types, 278
vias, 278
wires, 278
opening
cells, 229
files, 107
T-Cell, 380
optimization (SPR)
factor, 1951
in placement, 1951
in routing, 1950
optimize network, 1017
optimize network (LVS command-line option), 1036
optimizing design rule checking, 545
OR operation, 447, 452
orientation
LComp, 1813
origin, 186
absolute, 185
OTC routing (SPR), 1928, 1929
outlining an object when selected, 99
output
automorph class example, 1011
detailed trial matching, 1013
fragmented class, 1015
MEMS design example, 1020
parameter matching example, 1011
parsing, 1010
output file (LVS command-line option), 1036, 1037
output file types, 972
overlap, 1813
overlap capacitance (SPR), 1953
overlap design rules, 537
over-the-cell routing (SPR), 1928, 1929

P

pad cell abutment ports, 1938, 1964
pad cell mirror ports, 1938, 1967
pad cells, 1964
without bond pads, 1966
corner, 1966
Port text size
  in GDSII format, 163
  scaling, 332

Ports, 289
  definitions for global signals, 1967
  drawing, 289
  editing, 360
  editing in-place, 291
  legacy, 291
  mapping (SPR), 1924, 1925
  placement for automatic routing, 487
  port editing mode, 291
  pushing in
    to a port, 291
postiteration matching, 1025
PostScript files, 350
power pad cells, 1965
power ports, 1960
  in pad cells, 1964
  in standard cells, 1960
power rail width, 1935
power rails, 1961
power signal
  in SPR, 1924
power supply pads, 1966
prematch file, 1025
  LVS command-line option, 1037
  preiteration matching, 1025
prematch files
  input, 1014
  output, 1014
  resolving fragmentation of an automorph class, 1013
primitives, 215
printing, 115
  headers & legends, 117
  printer memory settings, 118
printing cell hierarchy, 244
Probing >
  Go To, 1024
  Highlight, 1029
probing verification results, 1024
process definition files, 209, 213
process steps
  etch, 209
  grow/deposit, 209
  implant/diffuse, 210
rendering
  color, 98
  layers, 93, 95
  modes, 94
  outlines, 99
  passes, 94, 97
  patterns, 97
reorienting objects, 326
resistor
  syntax in EXT file, 928
resistor arrays, 431
resistors
  shorting out, 984
resolving
  discrepancies, 1028
  fragmentation of an automorph class, 1013
    with prematch files, 1013
Power supply pads, 1966
prematch file, 1025
LVS command-line option, 1037
preiteration matching, 1025
prematch files
  input, 1014
  output, 1014
  resolving fragmentation of an automorph class, 1013
Primitives, 215
Printing, 115
  headers & legends, 117
  printer memory settings, 118
Printing cell hierarchy, 244
Probing>
  Go To, 1024
  Highlight, 1029
Probing verification results, 1024
Process definition files, 209, 213
Process steps
  Etch, 209
  Grow/deposit, 209
  Implant/diffuse, 210
Rendering
  Color, 98
  Layers, 93, 95
  Modes, 94
  Outlines, 99
  Passes, 94, 97
  Patterns, 97
Reorienting objects, 326
Resistor
  Syntax in EXT file, 928
Resistor arrays, 431
Resistors
  Shorting out, 984
Resolving
  Discrepancies, 1028
  Fragmentation of an automorph class, 1013
    With prematch files, 1013
fragmented classes, 1016
resolving verification errors, 1024, 1029
results viewing environment (RVE)
Calibre, 959
Rotate/Flip commands, 326
round style (wire ends and joins), 297
router
grid spacing settings, 483
routing
assisted, 398
clock signals, 1921
over-the-cell, 1928, 1929
three-layer, 1928
routing wires
in standard cells, 1962
in standard cells (SPR), 1962
routing with generated cells, 391
row base name, 1927
row crosser cells, 1950, 1963
row crosser ports, 1927, 1963, 1992
row indent ratio, 1950
row length, 1949
rule exceptions, 538
rule sets, 547
rule types, 535
rulers, 294
drawing, 293
editing, 294
temporary, 333
running
interpreted macros, 1071
LVS batch files, 1030

S
saving
cells, 230, 234
files, 109
images, 121
saving layout images, 121
schematic driven layout
command menu, 467
commands, 467, 469, 470
devices, adding, 476
devices, adding to a netlist, 472
devices, naming, 475, 476
devices, parameters, 476
flylines, 466
implicit model definitions, 493
instances, adding, 474, 475
instances, placement for floorplanning, 492
markers, 466
minimum grid spacing, 483
navigator, opening, 471
navigator, toolbar icons, 479
netlist format, 492
netlist, formats, 472
netlist, importing, 471
netlist, wirelist, 493
ports, adding, 475
ports, box, 489
ports, I/O, 484
ports, kine, 489
ports, on grid, 484
ports, point, 488
routing grid, 483, 484
steps, 462
subcircuits, adding, 474, 475
T-Cells in, 475
T-Cells, adding, 474, 476
T-Cells, naming, 476
T-Cells, parameters, 476
terminals, bulk or substrate, 472
view types, 465
wire end style, 486
wirelist, 493
zooming to routing objects, 466
SDF calculation, 1950
SDF format, 1938, 1974
selected objects
list and measurements of, 185
selection
box, 76
cycling, 312
deselecting objects, 313
edge, 312
explicit, 311
extending, 312
implicit, 312
of objects, 311
range, 85, 312
universal, 313
universal deselection, 314
selection box, 76
selectivity
setting, 311
Self-Intersecting Polygons, 282
Flag self intersections in DRC, 545
self-intersecting polygons, 147, 150
self-intersecting wires, 147, 150
separation into classes, 1023
series, elements in, 1026–1027
set mode, 96
Setup >
Application, 61
Editing, 65
General, 63
UPI, 71
Warnings, 68
Design, 80
Drawing, 88

Index

Technology, 81
Xref files, 88
Layers, 89, 91, 99, 103, 445
General, 91
Rendering, 90, 93, 104, 105
Palette, 60
Special Layers, 100, 130
Setup > Design > Vias, 396
setup files
  Laker, 56
  moving objects from layer to layer, 327
  Virtuoso, 56
setup files, replacing, 55
setup information
  design rule check, 533
Setup Layers
  General, 156
setup window, 976
  Device Parameters, 980
  Input, 977
  Options, 986
  Output, 978
  Performance, 988
shortcut keys
  writing to a configuration file, 67
showing
  insides, 202
  layers, 192
  objects, 189
shrink operation, 448
signal ports, 1965
  on core cells, 1919
  global, 1921, 1967
  illustrated, 1962
  in pad cells, 1965
  on padframes, 1920
  in standard cells, 1961
site cell for automatic placement, 501
slicing objects, 343
slicing wires, 303
snapping
  objects, 304
  perpendicular, 331
snapping cursor, 83
soft connections
  extracting, 923
  LVS detection of, 987
spacing design rules, 536
Special Layers Setup, 130
special standard cells, 1963
specialized elements, 1020
specifying
  design rules, 542
SPICE commands
  .END, 1003
  .GLOBAL, 1001
  .INCLUDE, 999
  .MALIAS, 493, 1000
  .MODEL, 999
  .OPTION, 1001
  .PARAM, 1002
  .SUBCKT, 998
SPICE file format, 994
  commands, 999
  comment lines, 1004
  device statements, 995
  parameter values, 1003
  restrictions, 1004
  statements, 999
  subcircuit definitions
    subcircuit definitions, 998
    subcircuit instances, 995, 997
SPICE OUTPUT properties, 1993
  example, 1994
SPR
  defined, 1915
  design files, 1916
  design flow, 1916
  design tips, 1918
  files required for, 1916
  initializing setup, 1926
  procedure outlined, 1916
  setup, 1923
  technology setup, 1916
SPR Core Setup
  Design Rules, 1930
  General, 1927
  Global Signals, 1934
  I/O Signals, 1936
  Layers, 1928
  Placement, 1932
  Power, 1935
SPR layout files, 1917
SPR Pad Route Setup, 1941
  Core Signals, 1946
  Design Rules, 1944
  General, 1943
  Layers, 1943
  Padframe Signals, 1947
SPR Padframe Setup
  General, 1937
  Layout, 1938
SPR Place and Route, 1948
  stacked vias, 94
  rendering, 94
  standard cells, 1960
  libraries, 1960
  special, 1963
  standard delay format files, 1974
  standard DRC
    reading and writing attributes, 50
Standard toolbar, 170
starting L-Edit, 45
state variables, 1813
status bar, 976
status bar (L-Edit), 184
straightening curved segments, 280
stretch ports
  T-Cell builder, 385
subcircuit
  syntax in EXT file, 932
subcircuit cell design, 1988
subcircuit connection ports
  illustrated, 1991
  syntax, 998
subcircuit recognition, 924, 1980, 1988
  activation, 1987
  cell design for, 1988
  connection ports, 1989
  connections, 1989
  cross ports, 1992
subcircuit recognition layer, 1986, 1988
subcircuit recognition polygons, 1988
  illustrated, 1990
subtract mode, 96
surround design rules, 536
symbols
  in CIF, 153

T

Tanner legacy purpose, 481
Tanner Place and Route format, 1969
T-Cell
  callbacks, 383
  code templates, 248
  code templates for, 374, 377
  creating arrays with, 376
  editing, 248, 374
  editing in-place, 381
  generating, 382
  generating from layout, 374
  hiding or showing, 380
  icons indicating, 380
  instancing, 249, 381, 382
  layer changing with, 389
  opening, 380
  parameters
  parameters, parameter types, 374, 376
  regenerating, 382
  stretching, 66
  updating, 249
T-Cell builder
  generating from layout, 384
  locating parameters, 391
  logical expressions, 390
  parameter, 387
  repeating elements, 388
  replicating cells, 386
  stretch ports, 385
T-Cell dll’s, 377
TCL commands, 363
tech.db, 47
tech.db files, 47
tech.tdb files, 47
technology reference files, 48
technology setup
  importing from Laker, 56
  importing from Virtuoso, 56
  in SPR, 1916
technology units, 80
temporary rulers, 333
text
  adding to a design file, 126
  adding to layout, 127, 128
  scaling, 127, 128
text editor
  keyword groups, 79
  modification conflicts, 78
  setup, 77
text size
  instances, 73
  rulers, 333
text window, 989
textual descriptions of objects, 185
textual editing, 351
three-layer routing, 1928
tie-to-ground cell, 1927, 1963
tie-to-power cell, 1927, 1963
toolbars, 975–976
  Alignment, 177
  Base Point, 175, 176, 177, 182, 183
  deleting, 178
  Drawing, 173
  Multi Grid, 177
  renaming, 178
  resetting to default display, 178
  Verification, 175
Tools
  Clear Error Layer, 949
  Clear Generated Layers, 461
  Cross-Section, 211
  DRC
    Setup Design Rules, 542
  Extract
    General, 1982
    Output, 1984
    Subcircuit, 1986
  Generate Layers, 459
  Macro, 1068
  SPR
Place and Route, 1948
Setup, 1923
Summary, 1917
Tools > DRC, 550
Tools > DRC Box, 550
Tools > Goto Device
device, 919
Tools > Macro, 531, 1068
Tools > Repeat Macro
repeat macro command, 531
top-level, 218
top-level cells, 218
topological matching, 1023
tori
drawing, 281
TPR format, 1969
tutorial, 1006–1009

Undo, 314
undoing operations, 314
ungrouping objects, 315
universal selection, 313
update display, 71, 1070, 1124–1125
UPI
function prototypes, 1071
header files, 1071
hot key definitions, 1071
include files, 1071
LComp interface, 1071
library files, 1071
UPI operating modes, 1069
user configuration file
saving to a text file, 67
user files, 61
user interface, 974–993
user-programmable interface, 1043

valid vias
modifying the list of, 49
.vdb files, 972
examples, 1010
Vdd pad cells, 1965
verification errors, 1024, 1029
verification options, 1031–1040
verification queue, 992–993
verification results
LVS, 1024
Verification toolbar, 175
verification window, 991
version
displaying, 169
version information
displaying, 165
vertex snapping, 306
vertices
limitations on, 145
removing, 305
vertices, adding, 332
via reduction
in SPR, 1950
via surrounds (SPR), 1931
via widths (SPR), 1931
vias
arrays, 399
creating arrays of, 400
generated, 391
locked, 392
View
by instance, 465
by net, 465
by unrouted segment, 465
View >
Design Navigator, 241, 242, 243
Display, 188
Exchange, 207
Goto, 207
Hierarchy Level >
Show one less level, 201
Show one more level, 201
View hierarchy level, 201
Home, 203
Insides, 202
Layers, 192
Layers >
Hide Generated, 461
Show Generated, 461
Objects, 188, 190
Pan >
Down, 204
Left, 204
Object Pan >
Left, 204
Right, 205
Pan By, 204
Right, 204
To Cell Edge, 205
To Selections, 204
Up, 204
Redraw, 188
Toolbars, 170
Zoom >
In, 203
Mouse, 206
Out, 203
To Selections, 204
Zoom By, 203
View > Selection Detail, 185
view types, 215
viewing
  layout hierarchy, 201
views
  exchanging, 207
  panning, 204
  pinned, 172
  zooming, 203
Virtuoso files
  exporting, 147
  importing, 56, 133
  importing for Assura, 830
voltage source device definitions, 984
VStretch, 376

W

warnings
  displaying
    See quiet mode
winding number, 284
window stretch editing, 331
Windows clipboard
  copying to, 328
wire
  wide, 399
Wire Utilities, 299
wire widths
  in core routing, 1931
wires
  adding sections, 299
  adding vertices to, 332
  CIF-compatible styles, 298, 373
  converting to DXF polylines, 151
  default styles, 298
  drawing, 281
  editing, 356
  end styles, 297
  extending, 301
  GDSII-compatible styles, 298, 373
  join styles, 297
  joining, 299
  merging, 345
  nibbling, 344
  self-intersecting, 147, 150
  slicing, 303
  terminology, 296
  utilities, 299
  width, 296
workgroup files, 61
WORLD_MAX, 1073
write CAP file, 1950, 1953, 1975
write SDF file, 1919, 1938, 1974

X

XST format, 213

Y

yes to all questions (LVS command-line option), 1040

Z

zero-volt voltage sources, 984
zoom/selection box, 76
zooming, 203
Credits

Software Development

Seyran Avanesyan    David Cardoze
Barry Dyne          Nikita Jorniak
Dan’l Leviton       Jeff Miller
Satoshi Ono         Massimo Sivilotti
Ken Van de Houten   Kurt Van Ness
Nicolas Williams

Quality Assurance

Judy Bergstresser   Luba Gromova
Lena Neo

Documentation

Judy Bergstresser   Barry Dyne
Nicolas Williams
END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively “Products”) between the company acquiring the Products (“Customer”), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity (“Mentor Graphics”). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties’ entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. ORDERS, FEES AND PAYMENT.

1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer’s appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (each an “Order”), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not those documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order or presented in any electronic portal or automated order management system, whether or not required to be electronically accepted, will not be effective unless agreed in writing and physically signed by an authorized representative of Customer and Mentor Graphics.

1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice. Except for taxes imposed on payments by State, Province or other local jurisdiction, any applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer’s sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer’s behalf, Customer shall be liable for payment orders placed by such third party in the event of default.

1.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics’ delivery of Software by electronic means is subject to Customer’s provision of both a primary and an alternate e-mail address.

2. GRANT OF LICENSE. The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data (“Software”) are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer’s internal business purposes; (c) the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee’s residence, an airport or hotel, provided that such employee’s primary place of employment is the site where the Software is authorized for use. Mentor Graphics’ standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) other services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer provides any feedback or requests any change, enhancement or corrections, Mentor Graphics shall have the exclusive right to make any changes or enhancements to the Software, at its sole discretion.

3. ESC SOFTWARE. If Customer purchases a license to use development or prototyping tools of Mentor Graphics’ Embedded Software Channel (“ESC”), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable
files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer’s compiled computer program, provided that Customer distributes these files only in conjunction with Customer’s compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics’ real-time operating systems or other embedded software products into Customer’s products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. BETA CODE.

4.1. Portions or all of certain Software may contain code for experimental testing and evaluation (which may be either alpha or beta, collectively “Beta Code”), which may not be used without Mentor Graphics’ explicit authorization. Upon Mentor Graphics’ authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. Mentor Graphics may choose, at its sole discretion, not to release Beta Code commercially in any form.

4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer’s use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer’s evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer’s feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. RESTRICTIONS ON USE.

5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer’s employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively “Files”), including without limitation files containing Standard Verification Rule Format (“SVRF”) and Tcl Verification Format (“TVF”) which are Mentor Graphics’ trade secret and proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Products or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.

5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer’s employees or on-site contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.

5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense, or otherwise transfer the Products, whether by operation of law or otherwise (“Attempted Transfer”), without Mentor Graphics’ prior written consent and payment of Mentor Graphics’ then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics’ prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics’ option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer’s permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. SUPPORT SERVICES. To the extent Customer purchases support services, Mentor Graphics will provide Customer with updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics’ then current End-User Support Terms located at http://supportnet.mentor.com/supportterms.

7. LIMITED WARRANTY.

7.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer’s requirements or that operation of Products will be uninterrupted or error free. The
warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification, improper installation or Customer is not in compliance with this Agreement. MENTOR GRAPHICS’ ENTIRE LIABILITY AND CUSTOMER’S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS’ OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED “AS IS.”

7.2. THE WARRANTIES SET FORTH IN THIS SECTION 7 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

8. LIMITATION OF LIABILITY. EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY. EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, IN NO EVENT SHALL MENTOR GRAPHICS’ OR ITS LICENSORS’ LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

9. HAZARDOUS APPLICATIONS. CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY (“HAZARDOUS APPLICATIONS”). EXCEPT TO THE EXTENT THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. INDEMNIFICATION. CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS’ FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. INFRINGEMENT.

11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to such action. Customer understands and agrees that as conditions to Mentor Graphics’ obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense: (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

11.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics’ licensors who do not provide such indemnification to Mentor Graphics’ customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

11.4. THIS SECTION 11 IS SUBJECT TO SECTION 8 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS, AND CUSTOMER’S SOLE AND EXCLUSIVE REMEDY, FOR DEFENSE, SETTLEMENT AND DAMAGES, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

12. TERMINATION AND EFFECT OF TERMINATION.

12.1. If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any
provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer’s obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

12.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer’s possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.

13. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States (“U.S.”) government agencies, which prohibit export, re-export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export or re-export Products in any manner without first obtaining all necessary approval from appropriate local and U.S. government agencies. If Customer wishes to disclose any information to Mentor Graphics that is subject to any U.S. or other applicable export restrictions, including without limitation the U.S. International Traffic in Arms Regulations (ITAR) or special controls under the Export Administration Regulations (EAR), Customer will notify Mentor Graphics personnel, in advance of each instance of disclosure, that such information is subject to such export restrictions.

14. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. The parties agree that all Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to U.S. FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. government or a U.S. government subcontractor is subject solely to the terms and conditions set forth in this Agreement, which shall supersede any conflicting terms or conditions in any government order document, except for provisions which are contrary to applicable mandatory federal laws.

15. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

16. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer’s normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer’s software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer’s compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics’ request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 16 shall survive the termination of this Agreement.

17. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the U.S. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, U.S., if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre (“SIAC”) to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. Nothing in this section shall restrict Mentor Graphics’ right to bring an action (including for example a motion for injunctive relief) against Customer in the jurisdiction where Customer’s place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

18. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

19. **MISCELLANEOUS.** This Agreement contains the parties’ entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.