Contents

Preface .......................................................................................................................... 11
Related Documents ..................................................................................................... 11
Typographic and Syntax Conventions .................................................................... 12
  Syntax Conventions .................................................................................................. 12
  Data Types .............................................................................................................. 13
  SKILL Syntax Examples .......................................................................................... 15

1

About the Technology File and Display Resource File .......................................... 16
Design Framework II Technology Data .................................................................... 17
  Files Containing Technology Data ........................................................................ 17
  How Technology Data Fits into and Is Used in the Design Flow ......................... 18
The Technology File .................................................................................................. 18
  Technology File Development and Usage .............................................................. 19
  Technology File Organization ............................................................................... 20
The Display Resource File .......................................................................................... 29
  Display Resource File Development and Usage .................................................... 29
  Display Resource File Organization ...................................................................... 31
How the Technology File and Display Resource File Work Together ..................... 32
Command Interpreter Window Pull-Down Menu Commands .................................. 33
  Technology File CIW Pull-Down Menu Commands ............................................... 34
  Display Resource File CIW Pull-Down Menu Commands ........................................ 35
# Technology File and Display Resource File User Guide

## Working in a Design Manager Environment

### 2 Creating a Technology File: Methods and General Guidelines

#### Methods of Initial ASCII File Creation

#### General Guidelines for Specifying Technology Data

#### Technology File Statements

- The Technology File Include Statement
- The Technology File Comment Statement

### 3 Creating a Technology File: Control, Layer, and Device Definitions

#### Controls

- Sample Controls Class
- Specifying Controls

#### Layer Definitions

- Sample Layer Definitions Class
- Specifying Layer Definitions

#### Devices

- Sample Devices Class
- Specifying Devices

### 4 Creating a Technology File: Generic Rules

#### Layer Rules

- Sample Layer Rules Class
- Specifying Layer Rules

#### Physical Rules

- Sample Physical Rules Class
- Specifying Physical Rules
Electrical Rules ........................................................................................................ 90
  Sample Electrical Rules Class ........................................................................... 92
  Specifying Electrical Rules .............................................................................. 92

5
Creating a Technology File: Application-Specific Rules .................................. 95
Virtuoso Layout Editor Rules ............................................................................... 96
  Sample Layout Editor Rules Class .................................................................. 97
  Specifying Layout Editor Rules ...................................................................... 97
Virtuoso XL Rules .................................................................................................. 99
  Sample Virtuoso XL Rules Class ................................................................... 100
  Specifying Virtuoso XL Rules ....................................................................... 100
Virtuoso Compactor Rules ................................................................................... 103
  Sample Compactor Rules Class .................................................................... 104
  Specifying Compactor Rules ........................................................................ 105
Virtuoso Layout Synthesizer (LAS) Rules .............................................................. 112
  Sample Layout Synthesizer Rules Class ......................................................... 113
  Specifying Layout Synthesizer Rules .............................................................. 114
Place and Route Rules .......................................................................................... 120
  Sample Place and Route Rules Class .............................................................. 121
  Specifying Place and Route Rules ................................................................ 124

6
Creating a Display Resource File ......................................................................... 142
Methods of Initial Display Resource File Creation ............................................ 143
Display Resource File Contents .......................................................................... 143
  What a Display Resource File Defines ......................................................... 143
  Sample Display Resource File ...................................................................... 144
Specifying Display Resources ............................................................................. 145
  Specifying Display Devices: drDefineDisplay() .......................................... 146
  Specifying Colors: drDefineColor() ............................................................... 147
  Specifying Stipple Patterns: drDefineStipple() ............................................. 148
  Specifying Line Styles: drDefineLineStyle() ................................................ 149
  Specifying Display Packets: drDefinePacket() ............................................. 150
  Specifying Display Packet Aliases: drDefinePacketAlias() .......................... 154
# 7
Preface for Use with a Design

- **Generating a Technology Library**
  - Compiling an ASCII Technology File into a Library
  - Creating a New Technology Library from an Existing Technology Library
- **Checking a Technology File for Conformance to Application Requirements**
- **Attaching a Technology Library to a Design**
  - Attaching a Technology Library to a Design Library
  - Attaching a Technology Library to a Design Cell or Cellview
  - Device Definitions Used by the Software When You Attach One Technology File After Another to a Design Library
- **Detaching a Technology Library from a Cell or Cellview**
- **Ensuring Desired Display Resource File Usage**

# 8
Editing, Reusing, and Merging Technology File Data

- **The Technology File Updating Process**
- **Methods for Editing a Technology File**
- **Reusing a Technology File or Library to Build a New Library**
  - Creating an ASCII File from a Binary Technology File
  - Copying a Technology Library to Use As a Basis for Creating a New Technology Library
- **Loading Technology Data into Virtual Memory**
  - Merging New Technology Data into an Existing Technology Library
  - Replacing an Existing Technology File in a Technology Library
- **Editing Class Data through the CIW Pull-Down Menus**
  - Class Data Accessible through the CIW Pull-Down Menus
  - The CIW Pull-Down Menus
  - Layer Browsers
- **Editing Class Data with SKILL Functions**
  - The Controls Class
  - The Layer Definitions Class
  - The Layer Rules Class
  - The Physical Rules Class
  - The Electrical Rules Class
9 Editing Class Data through CIW Pull-Down Menus: Controls and Layer Definitions

Editing Controls Class Data ................................................................. 207
Editing Layer Definitions Class Data ................................................. 210
  Adding a Layer Definition ................................................................. 210
  Editing Layer Display ........................................................................ 214
  Editing Layer Attributes ..................................................................... 216
  Renaming Layers or Purposes ............................................................... 218
  Changing Layer Priorities ................................................................... 220
  Adding or Editing Layer Properties ...................................................... 223
  Editing Multiple Layer-Purpose Pairs .................................................. 226
  Deleting Layers ................................................................................... 227
10 Editing Class Data through CIW Pull-Down Menus: Generic Rules

Editing Layer Rules Class Data ......................................................... 229
Editing Physical Rules Class Data ................................................. 236
Editing Electrical Rules Class Data ............................................. 240

11 Editing Class Data through CIW Pull-Down Menus:
Application-Specific Rules .......................................................... 244
Editing Layout Editor Rules Class Data ........................................ 245
Editing Virtuoso XL Rules Class Data .......................................... 248
Editing Virtuoso Compactor Rules Class Data .............................. 252
Editing Layout Synthesizer Rules Class Data ................................. 258

12 Editing, Reusing, and Merging Display Resources ............... 267
The Display Resource Updating Process ........................................ 268
Methods for Editing Display Resources ......................................... 269
Reusing a Display Resource File to Build a New Display Resource File ............................... 269
Merging Display Resource Files ........................................................ 270
  Merging Multiple Display Resource Files into One File .................. 271
  Merging New Display Resource Data into the Display Resource Data in Virtual Memory .... 272
Editing Display Resource Data through the CIW Pull-Down Menus .............................................. 274
  Display Resource Data Accessible through the CIW ................. 275
  Changing a Display Packet Definition ........................................ 275
  Changing a Layer Display .......................................................... 276
  Adding Color ........................................................................ 277
  Editing Color ........................................................................ 279
  Adding a Stipple Pattern .......................................................... 279
  Editing a Stipple Pattern .......................................................... 281
  Adding a Line Style ................................................................ 281
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editing a Line Style</td>
<td>282</td>
</tr>
<tr>
<td>Adding a Display Device</td>
<td>283</td>
</tr>
<tr>
<td>Deleting Display Resources</td>
<td>284</td>
</tr>
<tr>
<td>Deleting Colors, Stipples, and Line Styles</td>
<td>284</td>
</tr>
<tr>
<td>Deleting a Display Packet</td>
<td>289</td>
</tr>
<tr>
<td>Editing Display Resource Data with SKILL Functions</td>
<td>289</td>
</tr>
<tr>
<td>Reloading Source Display Resource Files</td>
<td>291</td>
</tr>
<tr>
<td>Saving Display Resource Data to a File</td>
<td>292</td>
</tr>
<tr>
<td>Testing a Display Resource File</td>
<td>294</td>
</tr>
<tr>
<td>Editing a Saved Display Resource File</td>
<td>295</td>
</tr>
<tr>
<td>About Save Changes Message Boxes</td>
<td>297</td>
</tr>
</tbody>
</table>

A

Form Descriptions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Technology Library Form</td>
<td>300</td>
</tr>
<tr>
<td>SKILL Function to Display Form</td>
<td>300</td>
</tr>
<tr>
<td>Check Technology File Form</td>
<td>301</td>
</tr>
<tr>
<td>SKILL Function to Display Form</td>
<td>301</td>
</tr>
<tr>
<td>Attach Technology Library to Design Library Form</td>
<td>302</td>
</tr>
<tr>
<td>SKILL Function to Display Form</td>
<td>302</td>
</tr>
<tr>
<td>Dump Technology File Form</td>
<td>303</td>
</tr>
<tr>
<td>SKILL Function to Display Form</td>
<td>303</td>
</tr>
<tr>
<td>Load Technology File Form</td>
<td>304</td>
</tr>
<tr>
<td>SKILL Functions to Display Form</td>
<td>304</td>
</tr>
<tr>
<td>Layer Purpose Pair Editor Form</td>
<td>305</td>
</tr>
<tr>
<td>Technology File – Technology File Set Up Form</td>
<td>307</td>
</tr>
<tr>
<td>SKILL Function to Display Form</td>
<td>307</td>
</tr>
<tr>
<td>Technology File – Layer Browser Forms</td>
<td>308</td>
</tr>
<tr>
<td>Discard Edits To Technology File Form</td>
<td>309</td>
</tr>
<tr>
<td>SKILL Function to Display Form</td>
<td>309</td>
</tr>
<tr>
<td>Save Technology File Form</td>
<td>310</td>
</tr>
<tr>
<td>SKILL Function to Display Form</td>
<td>310</td>
</tr>
</tbody>
</table>
## Technology File – Control Form

**Add Layer Purpose Pair Form** ................................................................. 312

**Add Purpose Form** ................................................................................. 314

**Edit Layer Purpose Pair Form** ............................................................. 315

**Rename Layer Form and Rename Purpose Form** .................................. 316
   - **Rename Layer Form** ........................................................................ 316
   - **Rename Purpose Form** ................................................................. 316

**Layer Property Editor Form** ................................................................. 317

**Add Property and Modify `<property_name>` Forms** ............................. 318

## Technology File – Layer Rules Form

**Technology File – Physical Rules Form** ............................................... 320

**Technology File – Electrical Rules Form** ............................................ 321

**Technology File – LE Rules (Lsw Layers) Form** ................................. 322

**Technology File – LX Rules Form** ....................................................... 323

**Technology File – Compactor Rules Form** ......................................... 324

**Technology File – LAS Rules Form** ..................................................... 327

## Merge Display Resource Files (DRF) Form

**Display Resource Editor (DRE) Form** .................................................. 330

**Fill Color Editor, Outline Editor, and Color Editor Forms** .................. 332

**Stipple Editor Form** ............................................................................. 333

**Line Style Editor Form** ........................................................................ 334

## New Device Form

**Load Form or Save Form** ..................................................................... 336

### B

**System-Reserved Layers and Purposes** .................................................. 337

**Layers** .................................................................................................. 338

**Purposes** .............................................................................................. 341

### C

**Resolving Layer Errors** ....................................................................... 343

**Adding a Layer to a Technology File** .................................................. 344
   - **Adding a Layer by Manually Editing the Technology File** ............... 344
   - **Adding a Layer Using Forms** ....................................................... 347
Resolving Inconsistent Layer Names .......................................................... 349
Fixing Layers That Do Not Appear Correctly ................................................ 351
  Editing a Display Packet ............................................................................. 351
  Selecting a Different Display Packet ......................................................... 351
  Creating a New Display Packet ................................................................. 351

D
Technology File and Display Resource File Examples ............................... 354
  Technology File Example ............................................................................ 355
  Display Resource File Example ................................................................. 361
Preface

In this manual, you’ll learn how to create and maintain a technology file and display resource file, both of which define the technology information you need to create and view your designs. This manual assumes you are familiar with the development and design of integrated circuits.

The preface discusses the following:

- Related Documents on page 11
- Typographic and Syntax Conventions on page 12

Related Documents

This manual includes some information about the applications that use the technology and display resource files, but you should check any specific details with the documentation for the applications you use.

The following documents give you more information about technology data, functions, and commands, and the applications that use them:

- For information about installing the product, see the Cadence® Installation Guide.
- For known problems and solutions for the technology file, see Technology File Known Problems and Solutions.
- For known problems and solutions for the display resource file, see Display Resource Editor Known Problems and Solutions.
- For information about new features, enhancements, and PCRs fixed in this release for the technology file, see the Technology File Product Notes.
- For information about new features, enhancements, and PCRs fixed in this release for the display resource file, see the Display Resource Editor Product Notes.
- For information about using Assura™ Diva® rules files, see the Assura Diva Verification Reference.
For additional information about the technology data used with the Virtuoso® layout editor, see “Using the Technology File” in the Virtuoso Layout Editor User Guide.

For additional information about the technology data used with Virtuoso XL, see “Editing Your Technology File for Virtuoso Layout Accelerator” in the Virtuoso Layout Accelerator User Guide.

For information about how the Virtuoso compactor uses symbolic data and DRC rules, see the Virtuoso Compactor Reference Manual, Chapter 1 and Appendix B.

For additional information about the technology data used with the Virtuoso layout synthesizer, see “Setting up the Technology File for LAS” in the Virtuoso Layout Synthesizer User Guide.

For information about the differences between LEF technology data and the technology file for the Preview Silicon Ensemble™ place-and-route software, see the Preview Gate Ensemble Reference, the LEF Data Map section of Chapter 4.

### Typographic and Syntax Conventions

The following sections explain the syntax conventions used in this document, including technology file syntax and Cadence® SKILL language syntax.

#### Syntax Conventions

This list describes the syntax conventions used in this document.

**literal (LITERAL)**

Nonitalic (UPPERCASE) words indicate keywords that you must enter literally. These keywords represent command (function, routine) or option names.

**argument (z_argument)**

Words in italics indicate user-defined arguments for which you must substitute a name or a value. (The characters before the underscore (_) in the word indicate the datatypes that this argument can take. Names are case sensitive. Do not type the datatype and underscore (z_) before your arguments.)

| Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character. |
Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices. You can choose one argument from the list.

{ }  Braces are used with OR-bars and enclose a list of choices. You must choose one argument from the list.

...  Three dots (....) indicate that you can repeat the previous argument. If you use them with brackets, you can specify zero or more arguments. If they are used without brackets, you must specify at least one argument, but you can specify more.

argument... ;specify at least one,
           ;but more are possible

[argument]... ;you can specify zero or more

,...  A comma and three dots together indicate that if you specify more than one argument, you must separate those arguments by commas.

=>  A right arrow points to the return values of the function. Variable values returned by the software are shown in italics. Returned literals, such as t and nil, are in plain text. The right arrow is also used in code examples in SKILL manuals.

/  A slash separates the possible values that can be returned by a SKILL function.

Note: The language requires any characters not included in the list above. You must enter required characters literally.

Data Types

The following table summarizes the data type prefixes and data types used in Cadence SKILL functions:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Internal Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>array</td>
<td>array</td>
</tr>
<tr>
<td>b</td>
<td>ddUserType</td>
<td>Boolean</td>
</tr>
<tr>
<td>c</td>
<td>opfcontext</td>
<td>OPF context</td>
</tr>
</tbody>
</table>
### Technology File and Display Resource File User Guide

**Preface**

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Internal Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>dbobject</td>
<td>Cadence database object (CDBA)</td>
</tr>
<tr>
<td>e</td>
<td>envobj</td>
<td>environment</td>
</tr>
<tr>
<td>f</td>
<td>flonum</td>
<td>floating-point number</td>
</tr>
<tr>
<td>F</td>
<td>opffile</td>
<td>OPF file ID</td>
</tr>
<tr>
<td>g</td>
<td>general</td>
<td>any data type</td>
</tr>
<tr>
<td>G</td>
<td>gdmSpecIlUserT ype</td>
<td>gdm spec</td>
</tr>
<tr>
<td>h</td>
<td>hdbobject</td>
<td>hierarchical database configuration object</td>
</tr>
<tr>
<td>l</td>
<td>list</td>
<td>linked list</td>
</tr>
<tr>
<td>m</td>
<td>nmpIlUserT ype</td>
<td>nmpIl user type</td>
</tr>
<tr>
<td>M</td>
<td>cdsEvalObject</td>
<td>—</td>
</tr>
<tr>
<td>n</td>
<td>number</td>
<td>integer or floating-point number</td>
</tr>
<tr>
<td>o</td>
<td>userType</td>
<td>user-defined type (other)</td>
</tr>
<tr>
<td>p</td>
<td>port</td>
<td>I/O port</td>
</tr>
<tr>
<td>q</td>
<td>gdmSpecListIlUserT ype</td>
<td>gdm spec list</td>
</tr>
<tr>
<td>r</td>
<td>defstruct</td>
<td>defstruct</td>
</tr>
<tr>
<td>R</td>
<td>rodObj</td>
<td>ROD object</td>
</tr>
<tr>
<td>s</td>
<td>symbol</td>
<td>symbol</td>
</tr>
<tr>
<td>S</td>
<td>stringSymbol</td>
<td>symbol or character string</td>
</tr>
<tr>
<td>t</td>
<td>string</td>
<td>character string (text)</td>
</tr>
<tr>
<td>u</td>
<td>function</td>
<td>function object, either the name of a function (symbol) or a lambda function body (list)</td>
</tr>
<tr>
<td>U</td>
<td>funobj</td>
<td>function object</td>
</tr>
<tr>
<td>v</td>
<td>hdbpath</td>
<td>—</td>
</tr>
<tr>
<td>w</td>
<td>wtype</td>
<td>window type</td>
</tr>
<tr>
<td>x</td>
<td>integer</td>
<td>integer number</td>
</tr>
<tr>
<td>y</td>
<td>binary</td>
<td>binary function</td>
</tr>
<tr>
<td>&amp;</td>
<td>pointer</td>
<td>pointer type</td>
</tr>
</tbody>
</table>
SKILL Syntax Examples

The following examples show typical syntax characters used in the technology file ASCII syntax and SKILL.

Example 1

list( g_arg1 [g_arg2] ...) => l_result

This example illustrates the following syntax characters.

list

Plain type indicates words that you must enter literally.

g_arg1

Words in italics indicate arguments for which you must substitute a name or a value.

( )

Parentheses separate names of functions from their arguments.

_  

An underscore separates an argument type (left) from an argument name (right).

[ ]

Brackets indicate that the enclosed argument is optional.

...

Three dots indicate that the preceding item can appear any number of times.

=>

A right arrow points to the description of the return value of the function. Also used in code examples in SKILL manuals.

l_result

All SKILL functions compute a data value known as the return value of the function.

Example 2

needNCells( s_cellType | st_userType x_cellCount) => t/nil

This example illustrates two additional syntax characters.

|  

Vertical bars separate a choice of required options.

/  

Slashes separate possible return values.
About the Technology File and Display Resource File

The chapter discusses the following:

- “Design Framework II Technology Data” on page 17
- “The Technology File” on page 18
- “The Display Resource File” on page 29
- “How the Technology File and Display Resource File Work Together” on page 32
- “Command Interpreter Window Pull-Down Menu Commands” on page 33
- “Working in a Design Manager Environment” on page 36
Design Framework II Technology Data

The Cadence® design framework II (DFII) technology data defines the parameters used in design sessions. The DFII tools use the technology data as you create your designs. The technology data includes layer definitions, device definitions, design rules, design application rules, display parameters, and plotter parameters—all of the information that defines the framework for creating designs.

Files Containing Technology Data

Most of the DFII technology data is distributed in two types of files, the technology file and the display resource file, as described in this User Guide. The Abstract Editor and Diva® products require their application-specific rules in separate files.

The technology file defines the materials and rules you use in your IC fabrication process. The technology file contains

- Layer definitions
- Device definitions
- Layer, physical, and electrical rules
- Rules specific to individual Cadence applications

The display resource file specifies how your layers appear on display devices. The display resource file contains

- Display device definitions
- Definitions of colors, stipple patterns, line styles, and fill styles
- Definitions of display packets, which are collections of colors, stipples, and line styles associated with particular display devices. A display packet specifies how you want a layer to be represented on the monitor or by a plotter. The technology file assigns a display packet to each layer it defines. In other words, a display resource file assigns a display packet to a display device or plotter, and the technology file assigns a display packet to each layer it defines.

The Diva rules files specify rules for Diva verification tools. For more information, refer to the Assura Diva Verification Reference.
How Technology Data Fits into and Is Used in the Design Flow

To run a DFII design session, you must define technology data in one or more technology files and display resource data in one or more display resource files. You then compile the technology file or files to create a technology library, which contains a binary technology file and device cellviews defined in the technology file or files. You must attach a technology library to your design library to apply the appropriate parameters and rules to your design; you can also attach technology libraries to specific cells or cellviews individually. When you run the design software, it uses the definitions in the attached technology library and display resource files to define your design.

The Technology File

This section introduces and presents an overview of technology file development and usage. It also summarizes technology file organization, presents definitions of the various kinds of data defined in a technology file, and summarizes which design software applications use which technology file data.
Technology File Development and Usage

The following illustrates the major steps for technology file development and usage:

1. **Create ASCII technology file**
   - Creating an ASCII technology file with a text editor, you define the technology data you want to use with a particular design. The ASCII technology file is a readable file that can contain comments and Cadence SKILL language routines in addition to the DFII technology data. You can include all of your technology data in one ASCII file or in several ASCII files that you compile into a single binary file.

2. **Compile ASCII technology file**
   - Compiling creates a binary technology file and generates the technology library defined by the technology file you compile (or multiple files you compile together). The technology library consists of the binary technology file (named `techfile.cds`) and the device cellviews defined in the ASCII technology file. **Note:** Do not rename the binary technology file or rearrange the file structure. Also, if you have rules files separate from the technology file, you must also file those in the technology library subdirectory.

3. **Check technology file for conformance to application requirements**
   - Checking ensures that the technology file data conforms to the requirements of the design applications with which it is to be used.

4. **Attach technology library to design library, cell, or cellview**
   - Attaching causes the design software to apply the technology library definitions and rules to the attached design library, cell, or cellview during a design session.

5. **Run a DFII design session**
   - When you open a design library, the DFII software automatically loads the attached technology library or libraries into virtual memory. During a design session, you can manipulate and edit the technology library in virtual memory; you can use the Command Interpreter Window (CIW) pull-down menus and SKILL functions to set and retrieve technology data.

Every DFII design uses a technology library. Usually, all of the designs in a design library use the same technology library, but DFII does not require that you follow this model. You can use any number of technology libraries in your design hierarchy; their number and contents depend upon your design requirements. Several design libraries can share the same technology library, or you can attach a different technology library to one or more of the designs in a library.
The following example shows a design library that is “attached to,” or uses, the technology file and the other technology data in a technology library. In this type of environment, you can share the same technology information among several design libraries.

![Diagram of Design Library and Technology Library](image)

**Technology File Organization**

The technology file is organized into classes and subclasses, also referred to as sections. A class is a category in which data with related functions is grouped. Each class begins with the class specifier (for example, `physicalRules`), followed by a parenthetical enclosure containing subclass specifications (for example, `orderedSpacingRules` and `spacingRules`), which define class data and rules.

```plaintext
physicalRules(
orderedSpacingRules(
    ;( rule layer1 layer2 value )
    ;( ---- ------ ------ ----- )
    ( minEnclosure "cellBoundary" nwell" 0.1 )
    ( minEnclosure "ndiff" "cont" 0.5 )
) ;end of Ordered Spacing Rules
)

spacingRules(
    ;( rule layer1 layer2 value )
    ;( ---- ------ ------ ----- )
    ( minSpacing "ndiff" 1.0 )
) ;end of Spacing Rules

);end of Physical Rules Class
```

The following section gives a brief description of the technology file classes and subclasses, plus a reference to detailed information on each. To see an example of an ASCII technology file, refer to Appendix D, “Technology File and Display Resource File Examples.”
Technology File Classes

The technology file contains the classes of information defined in this section.

The Controls class, Layer Definitions class, and Devices class are classes that provide definitions for design sessions.

**The Controls class (controls)** assigns values to parameters you can refer to in the rules you define and sets user read/write permissions on individual technology file classes. You can share parameters across rules classes. The individual subclasses

- Define parameters for use throughout the technology file
  (techParams)
- Define read/write permissions for technology file classes
  (techPermissions)

For more information about creating shared parameters, refer to “Controls” on page 43.

**The Layer Definitions class (layerDefinitions)** describes the layers you use in your designs. These subclasses

- Define the layers that can be used to define a layer-purpose pair
  (techLayers)
- Define the purposes that can be assigned to layer-purpose pairs
  (techPurposes)
- List layer-purpose pairs in priority order
  (techLayerPurposePriorities)
- Define the display attributes of a layer
  (techDisplays)
- Specify user-defined properties for specific layer-purpose pairs
  (techLayerProperties)

For more information about defining layers, refer to “Layer Definitions” on page 47.

**The Devices class (devices)** defines the devices you use with the Virtuoso® compactor and Virtuoso layout synthesizer software and the rule contact devices you use with Preview Gate Ensemble® and Preview Silicon Ensemble™ place-and-route software. You can also create user-defined devices in the Devices class. The individual subclasses

- Activate all Cadence-predefined device types
  (tcCreateCDSDeviceClass) and declare Cadence-predefined
About the Technology File and Display Resource File

- Contact devices
  (symContactDevice and ruleContactDevice)

- Enhancement devices
  (symEnhancementDevice)

- Depletion devices
  (symDepletionDevice)

- Pin devices
  (symPinDevice)

- Rectangular pin devices
  (symRectPinDevice)

- Define customer, or user-defined, device types
  (tcCreateDeviceClass)

- Declare devices of any type
  (tcDeclareDevice)

For more information about creating devices in the technology file, refer to “Devices” on page 56.

Generic rules classes apply across design applications and define design rules and constraints. The Layer Rules, Physical Rules, and Electrical Rules classes are generic rules classes.

The Layer Rules class (layerRules) identifies layers that can be used as vias to conduct between routing layers and layers that are equivalent to each other. It also defines the Stream translation data required for translating designs from Stream format. The individual subclasses

- Define layers that conduct between two other layers
  (viaLayers)

- List layer-purpose pairs that represent the same type of material
  (equivalentLayers)

- List Stream translation data for layer-purpose pairs
  (streamLayers)

For more information about layer rules, refer to “Layer Rules” on page 78.

The Physical Rules class (physicalRules) defines the physical parameters of layers in your layout design. Physical rules include layer spacing, overlap, and equivalence rules. The individual subclasses

April 2001  22  Product Version 4.4.6
List spacing rules in which the order of layers is not important \( (\text{spacingRules}) \)

List spacing rules in which the order of layers is important \( (\text{orderedSpacingRules}) \)

Specify a multiple for grid snapping
\( (\text{mfgGridResolution}) \)

For more information about creating physical rules, refer to “Physical Rules” on page 83.

The Electrical Rules class \( (\text{electricalRules}) \) defines the electrical characteristics of the layers in your design. Electrical characterization rules define resistance, capacitance, and current density. The individual subclasses

List characterization rules in which the order of layers is not important
\( (\text{characterizationRules}) \)

List characterization rules in which the order of layers is important
\( (\text{orderedCharacterizationRules}) \)

For more information about creating electrical rules, refer to “Electrical Rules” on page 90.

Application-specific rules classes apply only to specific design applications. They define rules that apply to specific DFFI physical design applications (Virtuoso layout editor, layout accelerator, layout synthesizer, and compactor) and place-and-route applications (such as Preview Gate Ensemble and Preview Silicon Ensemble).

The Layout Editor Rules class \( (\text{leRules}) \) specifies rules for the Virtuoso layout editor physical design application. Its subclass

Specifies the order in which layers are displayed in the Layer Selection Window (LSW)
\( (\text{leLswLayers}) \)

For more information about specifying layout editor rules, refer to “Virtuoso Layout Editor Rules” on page 96.

The Virtuoso XL Rules class \( (\text{lxRules}) \) specifies rules for the Virtuoso layout accelerator (Virtuoso XL) physical design application. The individual subclasses

List the layers to be monitored by the online extractor
\( (\text{lxExtractLayers}) \)

List the layers that cannot overlap in a Virtuoso XL design \( (\text{lxNoOverlapLayers}) \)

Define templates for relative object design (ROD) multipart paths (MPPs)
\( (\text{lxMPPTemplates}) \)

For more information about specifying Virtuoso XL rules, refer to “Virtuoso XL Rules” on page 99.
The Virtuoso Compactor Rules class (`compactorRules`) specifies rules for the Virtuoso compactor design application. The individual subclasses

- Specify how the compactor is to use specified layers (`compactorLayers`)
- Define the wires used in a design (`symWires`)
- Define design rules followed by the compactor (`symRules`)

For more information about specifying Virtuoso compactor rules, refer to “Virtuoso Compactor Rules” on page 103.

The Layout Synthesizer Rules class (`lasRules`) specifies rules for the Virtuoso layout synthesizer (LAS) design application. The individual subclasses

- Specify how LAS uses layers when it automatically synthesizes a design (`lasLayers`)
- Specify cells used as devices during design synthesis (`lasDevices`)
- Specify wires and how they are used during design synthesis (`lasWires`)
- Specify properties that control how LAS synthesizes a design (`lasProperties`)

For more information about specifying layout synthesizer rules, refer to “Virtuoso Layout Synthesizer (LAS) Rules” on page 112.

The Place and Route Rules class (`prRules`) specifies rules for the place- and-route applications (such as Preview Gate Ensemble and Preview Silicon Ensemble). The individual subclasses

- Define the routing direction of layers used for routing (`prRoutingLayers`)
- Define the default and nondefault vias to be used in routing (`prViaTypes`)
- Specify the via layers that can be stacked (`prStackVias`)
- Define master slice layers (`prMastersliceLayers`)
- Define the rules for placing vias (`prViaRules`)
Define the rules for generating vias 
\( \text{prGenViaRules} \)

Define the rules for placing turn vias 
\( \text{prTurnViaRules} \)

Define the rules for routing with nondefault wire widths \( \text{prNonDefaultRules} \)

Specify minimum allowable spacing between two regular geometries on different nets 
\( \text{prRoutingPitch} \)

Define the distance between the placement grid and the routing grid when there is a routing grid between two placement grids 
\( \text{prRoutingOffset} \)

Define the overlap layer or layers used to display the overlap boundary 
\( \text{prOverlapLayer} \)

For more information, refer to “Place and Route Rules” on page 120.

Technology File–to–Application Map

The technology file is organized functionally, with the data required for different applications distributed in the various technology file classes. The table below shows which classes and subclasses (sections) of the technology file are used by which DFII applications, as identified in the table by the following numbers:

1. Virtuoso layout editor
2. Virtuoso layout accelerator
3. Virtuoso compactor
4. Virtuoso layout synthesizer
5. Preview Silicon Ensemble
6. Preview Gate Ensemble
7. other DFII applications
Technology File Class or Subclass | Application
---|---
controls() | The Controls class sets parameters that can be used throughout the technology file.
  - techParams()  
  - techPermissions()  

layerDefinitions() | Layer properties are user defined.
  - techLayers() x x x x x x x x
  - techPurposes() x x x x x x
  - techLayerPurposePriorities() x x x x x x
  - techDisplays() x x x x x x
  - techLayerProperties()  

devices() |  
  - tcCreateCDSDeviceClass() x x x x x x
  - symContactDevice() x x x x x x
  - ruleContactDevice() x x
  - symEnhancementDevice() x x
  - symDepletionDevice() x x
  - symPinDevice() x x x x x x
  - symRectPinDevice() x x x
  - tcCreateDeviceClass() x x x x x
  - tcDeclareDevice() x x x x x

layerRules() |  
  - viaLayers() x x x x
  - equivalentLayers() x x x
### Technology File Class or Subclass

<table>
<thead>
<tr>
<th>Technology File Class or Subclass</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>streamLayers()</strong></td>
<td>To translate designs with the pipo translator, use streamLayers.</td>
</tr>
<tr>
<td><strong>physicalRules()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>spacingRules()</strong></td>
<td>x x x x</td>
</tr>
<tr>
<td><strong>orderedSpacingRules()</strong></td>
<td>x x</td>
</tr>
<tr>
<td><strong>mfgGridResolution()</strong></td>
<td>x x x x</td>
</tr>
<tr>
<td><strong>electricalRules()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>characterizationRules()</strong></td>
<td>x x x x x x</td>
</tr>
<tr>
<td><strong>orderedCharacterizationRules()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>leRules()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>leLswLayers()</strong></td>
<td>x x x x</td>
</tr>
<tr>
<td><strong>lxRules()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>lxExtractLayers()</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>lxNoOverlapLayers()</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>lxMPPTemplates()</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>compactorRules()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>compactorLayers()</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>symWires()</strong></td>
<td>x x</td>
</tr>
<tr>
<td><strong>symRules()</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>lasRules()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>lasLayers()</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>lasDevices()</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>lasWires()</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>lasProperties()</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>prRules()</strong></td>
<td></td>
</tr>
<tr>
<td><strong>prRoutingLayers()</strong></td>
<td>x x</td>
</tr>
<tr>
<td>Technology File Class or Subclass</td>
<td>Application</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>prViaTypes()</td>
<td></td>
</tr>
<tr>
<td>prStackVias()</td>
<td></td>
</tr>
<tr>
<td>prMastersliceLayers()</td>
<td></td>
</tr>
<tr>
<td>prViaRules()</td>
<td></td>
</tr>
<tr>
<td>prGenViaRules()</td>
<td></td>
</tr>
<tr>
<td>prTurnViaRules()</td>
<td></td>
</tr>
<tr>
<td>prNonDefaultRules()</td>
<td></td>
</tr>
<tr>
<td>prRoutingPitch()</td>
<td></td>
</tr>
<tr>
<td>prRoutingOffset()</td>
<td></td>
</tr>
<tr>
<td>prOverlapLayer()</td>
<td></td>
</tr>
</tbody>
</table>
The Display Resource File

This section introduces and presents an overview of display resource file development and usage. It also summarizes display resource file organization and presents definitions of the various kinds of data defined in a display resource file.

Display Resource File Development and Usage

The following illustrates the major steps for display resource file development and usage:

1. **Create an ASCII display resource file**
   - Creating an ASCII display resource file with a text editor, you define the display data you want to use with specific display devices. The display resource file groups display data in display packets that it assigns to display devices. You can have multiple display resource files filed in various locations. Each, however, must be named `display.drf`.

2. **Select a location for the display resource file**
   - When selecting where to file your `display.drf` file, you must consider the fact that the DFII software loads and merges up to six display resource files from predefined locations. See “How the Design Framework II Software Handles Multiple Display Resource Files” on page 29 for details.

3. **File the display resource file in the selected location**

4. **Run a Cadence DFII design session**
   - As mentioned above, when you open a design library, the DFII software automatically loads up to six display resource files into virtual memory. During a design session, you can manipulate and edit the display resource data in virtual memory with the Display Resource Editor or SKILL functions.

How the Design Framework II Software Handles Multiple Display Resource Files

The DFII software uses a display resource file that it creates in virtual memory at startup. This display resource file is a blend of data from as many as six `display.drf` files. Because the
files are merged in sequence, files loaded later in the sequence can redefine display packets, colors, line styles, stipples, and display devices defined by files loaded earlier.

The following are the source display resource files listed in the order in which they are loaded:

- The Cadence-supplied default display resource file
  
  `install_dir/share/cdssetup/dfII/default.drf`

  This file is used with the Composer™ schematic capture application.

- A local display resource file you specify using the `drfPath` variable in your `.cdsenv` file. The syntax is

  `graphic drfPath string "path/display.drf"`

  This is an optional file you can use to provide required display resource definitions. Naming the file `display.drf` is recommended but not required.

- Optional site and project display resource files

  These are optional files your system administrator can place in the site and project directories, if those directories are set up at your site. These files must be called `display.drf`.

  For more information about these directories, refer to the *Cadence Application Infrastructure User Guide*.

- Personal display resource file

  `~/display.drf`

  This is an optional file that you can customize and place in your home directory. This file must also be called `display.drf`.

- The current directory

  `./display.drf`

  This is an optional file that you can customize and place in the directory from which you start the software. This file must be called `display.drf`.

**Planning Display Resource File Updates for Proper Merging**

Because the system merges several files to create the display resource data you use to create your designs, you will need to plan updates to the data. There will be times when you will use the Display Resource Editor and save your changes to a new `display.drf` file. There will be other times when you will need to edit a source display resource file in a text
Display Resource File Organization

A display resource file is organized into sections that define display resources as described in the following paragraphs. To see an example of a display resource file (display.drf), refer to Appendix D, “Technology File and Display Resource File Examples.”

The display devices section (drDefineDisplay) lists the names of the display devices for which display information is defined in the display resource file.

For more information, refer to “Specifying Display Devices: drDefineDisplay()” on page 146.

The color definitions section (drDefineColor) defines the colors used with various display devices. This section applies specific color definitions to color names and associates them with specific display devices.

For more information, refer to “Specifying Colors: drDefineColor()” on page 147.

The stipple definitions section (drDefineStipple) defines the stipple patterns used with various display devices. This section applies specific stipple pattern bitmaps to stipple names and associates them with specific display devices.

For more information, refer to “Specifying Stipple Patterns: drDefineStipple()” on page 148.

The line style definitions section (drDefineLineStyle) defines the line styles used with various display devices. This section applies specific line style sizes and patterns to line style names and associates them with specific display devices.

For more information, refer to “Specifying Line Styles: drDefineLineStyle()” on page 149.

The display packet definitions section (drDefinePacket) defines the display packets used with various display devices. This section applies specific stipple patterns, line styles, fill colors, outline colors, and fill styles to display packet names and associates them with specific display devices.

For more information, refer to “Specifying Display Packets: drDefinePacket()” on page 150.

The display packet alias definitions section (drDefinePacketAlias) applies alias names to display packet names and associates them with specific display devices.

For more information, refer to “Specifying Display Packet Aliases: drDefinePacketAlias()” on page 154.
How the Technology File and Display Resource File Work Together

The technology file and display resource file together tell the design software how to display each layer on a specific display device. The technology file assigns a display packet, by name, to each layer. The display resource file assigns a display packet definition, with a display packet name, to each display device. To determine how to display a layer on a specific display device, the design software does the following:

■ First, in the technology file, the design software finds the name of the display packet assigned to the layer.

■ Then, in the display resource file, the design software finds the definition of the display packet by that name that is assigned to the display device in use.
As the sample illustrates, the display packet assignments in the two files:

**ASCII Technology File** defines layer-purpose pairs and assigns a display packet, by name, to a layer-purpose pair.

```
layerDefinitions(
  .
  .
  .
)
techDisplays(
  ;(LayerName Purpose Packet ...)
  (nwell net yelsilverdots_S . ...)
  .
  .
  .
)
```

**Display Resource File** defines display resources and packets and assigns a display packet, by name and definition, to a display device.

```
...#DefineDisplay(
  ;(DisplayName #Colors #Stipple #LineStyles)
  (display 52 32 32 )
  ...)
#DefineColor(
  ;(DisplayName ColorsName Red Green Blue)
  (display yel 255 255 0)
  (display silver 217 230 255)
  ...)
#DefineStipple(
  ;(DisplayName StippleName Bitmap)
  (display dots ( 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0))
  ...)
#DefineLineStyle(
  ;(DisplayName LineStyle Size Pattern)
  (display solid 1 (1 1 1))
  ...)
#DefinePacket(
  ;(DisplayName PacketName Stipple LineStyle Fill Outline)
  ( display yelsilverdots_S dots solid yel silver)
  ...)
...```

**Command Interpreter Window Pull-Down Menu Commands**

The Command Interpreter Window (CIW) provides pull-down menu commands that allow you to manipulate technology data and display resource data in virtual memory during a design session.
Technology File CIW Pull-Down Menu Commands

The technology file commands let you compile, dump, and edit technology data. The Technology File menu is on the CIW menu banner. This section introduces the Technology File commands. For detailed information about using these commands, refer to “Editing Class Data through the CIW Pull-Down Menus” on page 178.

*New* creates a new technology library by compiling an ASCII technology file or copying an existing binary technology library. It also loads the technology library into virtual memory.

*Load* compiles an ASCII technology file into an existing library and loads it into virtual memory.

*Dump* writes a technology file in virtual memory to an ASCII file and opens the file in an editor window for you to view and edit.

*Discard* deletes the current technology file from virtual memory and reloads technology data to virtual memory from disk.

*Check* verifies the rules in an ASCII technology file for a specific application.
Save writes a technology file in virtual memory to the binary file on disk.

Attach To assigns a technology file to a library, a cell, or a cellview.

Edit Layers lets you update the data in the Layer Definitions class of a technology file.

Set Up lets you edit various technology file classes and subclasses.

Display Resource File CIW Pull-Down Menu Commands

The Display Resource Editor (DRE) is a tool you can use to update the display resource file loaded into memory. You can create new colors, stipple patterns, and line styles, and you can modify the definitions of display packets. You can save the contents of memory to an ASCII file, and you can load ASCII files into memory.

You start the DRE from the CIW by choosing Tools – Display Resources – Editor. For information about using the DRE to edit your display packet definitions, refer to Chapter 12, “Editing, Reusing, and Merging Display Resources.”

Display Resources – Editor invokes the Display Resource Editor.
Display Resources – Merge Files merges multiple display resource files into a single display resource file.

Working in a Design Manager Environment

If you use a design manager, the system attempts to check the technology files out and in as you edit the layers. If you have your environment set to prompt you to check out or check in all or views, the system prompts you with check-out and check-in forms.

The system checks out the technology library when you
- Start the Edit Layers command and the technology library shown in the Technology Library cyclic field is not checked out
- Select another technology library that is not checked out

For more information about setting check-out options, refer to the Library Manager User Guide.

The system checks in the technology library when you
- Select another technology library and the library you just edited is checked out
- Quit the Layer Purpose Pair Editor

For more information about setting check-in options, refer to the Library Manager User Guide.
Creating a Technology File: Methods and General Guidelines

This chapter discusses the following:

- “Methods of Initial ASCII File Creation” on page 38
- “General Guidelines for Specifying Technology Data” on page 38
- “Technology File Statements” on page 39
Methods of Initial ASCII File Creation

You can create a new ASCII technology file by any of the following methods:

■ In a text editor, create a technology file from scratch
■ Copy a sample ASCII technology file from the Cadence® installation and edit it in a text editor to produce your own technology file
■ Copy an existing ASCII technology file from your company’s files and edit it in a text editor to produce your own ASCII technology file
■ Dump a technology file from an existing technology library and edit it in a text editor to produce your own ASCII technology file

Whatever method you use, the structure of and requirements for specifying the technology file classes and subclasses remain the same. Chapters 3 through 5 define how to specify technology data.

■ Chapter 3 defines the rules and guidelines for specifying data in the Controls, Layer Definitions, and Devices classes and their subclasses.
■ Chapter 4 defines the rules and guidelines for specifying data in the Layer Rules, Physical Rules, and Electrical Rules classes and their subclasses.
■ Chapter 5 defines the rules and guidelines for specifying data in the Layout Editor Rules, Virtuoso® XL Rules, Virtuoso Compactor Rules, Layout Synthesizer Rules, and Place and Route Rules classes and their subclasses.

General Guidelines for Specifying Technology Data

The following are some guidelines for specifying technology file class and subclass data:

■ You must supply all arguments unless they are identified as optional by being shown in square brackets ([ ]) in the syntax specifications.
■ You must specify at least one argument, but can specify more, for arguments that are followed by three dots (... ) in the syntax specifications.
■ You must specify keywords exactly as shown.
■ You can specify an expression for any user-defined argument.
■ For any layer argument, you can specify either a layer name or a specific layer-purpose pair. If you specify a layer-purpose pair, the value is that specific layer-purpose pair. If you specify a layer name, it implies all layer-purpose pairs that include that layer.
You must specify the appropriate data type for any user-defined argument. In the syntax specifications, the argument prefix (or characters before the underscore (_) in the argument name) indicate the data type. The data types used in technology file and display resource file SKILL syntax are as follows:

<table>
<thead>
<tr>
<th>Argument Prefix</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Cadence database object</td>
</tr>
<tr>
<td>g</td>
<td>general (any data type)</td>
</tr>
<tr>
<td>l</td>
<td>linked list</td>
</tr>
<tr>
<td>n</td>
<td>integer or floating-point number</td>
</tr>
<tr>
<td>s</td>
<td>symbol</td>
</tr>
<tr>
<td>t</td>
<td>character string (text)</td>
</tr>
<tr>
<td>x</td>
<td>integer number</td>
</tr>
</tbody>
</table>

For a complete list of data types supported by the Cadence SKILL language, see the *SKILL Language User Guide*.

**Technology File Statements**

The technology file can contain two statements that provide flexibility in technology file development and maintenance rather than providing class data. These are the following:

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>include</td>
<td>Includes another file in the current technology file</td>
</tr>
<tr>
<td>comment</td>
<td>Adds a comment that is preserved during technology file compilation and dumping</td>
</tr>
</tbody>
</table>

**The Technology File Include Statement**

In an ASCII technology file, you can include another file defining technology data by specifying an `include` statement. For example, assume that your technology data contains extensive device definitions. For ease of file maintenance, you might want to create a separate file containing the `devices` class data for your technology file. The `include`
statement enables you to put this data in a separate technology file and reference that file from your main technology file.

The syntax for a technology file include statement is as follows:

```
include("t_techFileName")
```

where:

- `t_techFileName` is the name of the file to include.

The following is an example of an `include` statement to include a file containing device definitions:

```
include("/usr1/smith/devices.def")
```

This statement, placed in the technology file where the `devices` class belongs, includes the `devices.def` file from the location `/usr1/smith` in the technology file. When the compiler encounters this statement, it retrieves and compiles the `devices.def` file as part of the current technology library.

**Note:** If you compile a technology file containing an include statement and then dump the ASCII technology file from the resultant technology library, the dumped technology file will not contain the include statement, but will contain the included technology file data instead.

**The Technology File Comment Statement**

You can add comments to a technology file by preceding them with a semicolon (;). However, comments added in this way are not preserved when you compile a technology file into a technology library and later dump an ASCII technology file from a technology library. The `comment` statement allows you to add comments that are preserved throughout compilation and subsequent technology file dumping. Comment statements must be made at the class level; they cannot be within the parentheses of a class or subclass. A comment statement applies to the class that immediately follows it.

The syntax for a technology file comment is as follows:

```
comment ( "t_comment"

)
```

where:

- `t_comment` is the comment text, which must be enclosed in quotation marks inside the parentheses of the comment statement
The following is an example of a comment statement:

```plaintext
comment (  
    "This comment applies to the Controls class."  
)  

controls (  
    techParams (  
        (theta 2.0)  
        (lambda 4.0)  
    )  
)
```

When you compile the technology file containing these statements into a technology library, the software assigns the comment to the class immediately following it; in this case, the `controls` class. If you subsequently dump the technology data from this library to an ASCII file, the software preserves the comment along with the `controls` class data. The other comments, identified by semicolons, are not preserved.
Creating a Technology File: Control, Layer, and Device Definitions

This chapter discusses the technology file classes that specify the following:

- “Controls” on page 43
- “Layer Definitions” on page 47
- “Devices” on page 56

Other technology file classes define the following:

- Layer rules, described in Chapter 4, “Creating a Technology File: Generic Rules.”
- Physical rules, described in Chapter 4, “Creating a Technology File: Generic Rules.”
- Electrical rules, described in Chapter 4, “Creating a Technology File: Generic Rules.”
- Application-specific rules, described in Chapter 5, “Creating a Technology File: Application-Specific Rules.”
**Controls**

Technology file controls allow you to

- Establish and assign values to parameters for use throughout a design session
- Assign read and write permissions to specific classes within the technology file

**Sample Controls Class**

The following sample Controls class illustrates the class and its subclasses (sections), along with the technology file controls they define.

For more information about the Controls class, refer to the *Technology File and Display Resource File SKILL Reference Manual*.

```plaintext
controls(
  techParams(
    ;( ParamName Value )
    ( lambda 0.3 )
    ( theta 2.0 )
    ( yx 3.2 )
    ...
    ( spd 0.01 )
  ); end of techParams

  techPermissions(
    ;( Class R Only R/W )
    ( devices ("doug" "mary") ("jo") )
    ( leRules ("joe" "jrm") ("lmh") )
    ...
    ( lasRules ("jrm") ("lmh") )
  ); end of techPermissions

); end of controls
```

**Technology Parameters subclass.** Specifies parameters and assigns values. Data for each parameter must be enclosed in parentheses. All parameter specifications must be enclosed within the parentheses of the subclass enclosure.

**Technology Permissions subclass.** Assigns read-only and read/write permissions to technology file classes. Data for each class must be enclosed in parentheses. All permission assignments must be enclosed within the parentheses of the subclass enclosure.
Specifying Controls

Technology file controls are optional. As illustrated in the sample Controls class, the class must be specified by enclosing the subclass definitions within the `controls()` class enclosure. The following paragraphs provide detailed information about specifying subclass data.

Setting Parameters: `techParams()`

To share data across multiple technology file classes, set and define parameters in the `controls()` class of the technology file and then specify those parameters as needed throughout the technology file.

The `techParams()` subclass (section) of the Controls class defines the parameters to be used in your technology file. In this subclass, you can specify and assign a value to a parameter, then use that parameter throughout the technology file instead of specifying the value. Whenever the parameter is encountered, the software evaluates it to the value currently assigned to it. If you need to change that value, you change it only once, in the parameter definition, rather than changing every place the value is used throughout the technology file.

**Technology Parameters subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Parameter name.** Each parameter name must be unique. If not, parameters of the same name specified later in the list override those specified before them.
- **Parameter value.** Specify the value for the software to use whenever the parameter is encountered.

```
techParams(
    ;( paramName Value )
    ( lambda 0.3 )
    ( theta 2.0 )
    ( yx 3.2 )
    ...
    ...
    ( spd 0.01 )
) ; end of techParams
```
Using Parameters in your Technology File

Once a parameter is defined in the techParams subclass, you can invoke that parameter whenever needed in your technology file by specifying it with the following syntax:

\[ \text{techParam ( paramName )} \]

The following example sets a parameter (\( \text{theta} \)) to a value (2.0) and uses that parameter in design rules from two classes:

```plaintext
controls(
    techParams(
        (theta 2.0)
    )
)
physicalRules(
    spacingRules(
        (minWidth metal1 techParam("theta") * 2)
    )
)
electricalRules(
    characterizationRules(
        (currentDensity metal1 techParam("theta") )
    )
)
```

During technology file compilation, the software stores the expression \( \text{techParam("theta")} \times 2 \) as the rule value for the \( \text{minWidth} \) spacing rule and the expression \( \text{techParam("theta")} \) as the rule value for the \( \text{currentDensity} \) characterization rule; it evaluates each expression when it is accessed, as follows:

\[
\begin{align*}
\text{minWidth} &= 2.0 \times 2 = 4.0 \\
\text{currentDensity} &= 2.0
\end{align*}
\]

If, at a later time, you want to change the parameter value from 2.0 to any other value, you can do so by changing the techParams specification. You can change a parameter value in any of the following ways:

■ directly in the ASCII technology file so that when you recompile the technology library, the new value is assigned to the parameter

■ in virtual memory during a design session with the techSetParam SKILL function

■ in virtual memory from the CIW by accessing the Technology File – Control form

For more information about the techParams() subclass, refer to the Technology File and Display Resource File SKILL Reference Manual.
Setting Permissions: techPermissions()

You can set read-only and read/write permissions on classes in your technology file for different system users. Users can then access technology file classes only as their assigned permissions allow.

Note: If you do not set permissions for a class, then all users have read/write access to that class.

The techPermissions() subclass (section) of the Controls class sets the permissions for classes in your technology file.

**Technology Parameters subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Technology File Class.** Specify the name of the class to which to apply permissions.
- **Read-only permissions.** Specify the user names of the users to whom to assign read-only permission for the class.
- **Read/write permissions.** Specify the user names of the users to whom to assign read/write permission for the class.

```
techPermissions(
    ; ( Class R Only R/W )
    ( devices ("doug" "mary") ("jo") )
    ( leRules ("joe" "jrm") ("lmh") )
    .
    ( lasRules ("jim") ("lmh") )
)
```

For more information about the techPermissions() subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual.*
Layer Definitions

In Cadence® design framework II (DFII) applications, you create devices using shapes that you create with the Virtuoso® layout editor. You create IC designs by placing instances of the devices with other shapes in a cellview. Each shape is displayed or plotted using a layer.

Each layer is defined by a layer-purpose pair, which consists of a unique layer name and purpose combination. The layer name usually indicates a type of manufacturing material. The purpose indicates the use of the layer or material. You can create multiple layers with the same name but different purposes.

A layer definition also includes a display packet and display attributes. The display packet determines how the layer appears on your monitor and plotting devices. The display attributes control how objects behave during editing and translation.

The Layer Definitions class contains information about the layers you use to create your DFII designs.

Sample Layer Definitions Class

The following sample Layer Definitions class illustrates the class and its subclasses (sections), along with the layer characteristics they define.

For more information about the Layer Definitions class, refer to the Technology File and Display Resource File SKILL Reference Manual.
LayerDefinitions()

techLayers(
    ;( layerName Layer# Abbreviation )
    ;User-Defined Layers:
    ;System-Reserved Layers:
    ( Unrouted 200 Unroute )
    ( Row 201 Row )
    ( Group 202 Group )
    .
    .
    ( background 254 bkground )
) ;end of techLayers

techPurposes(
    ;( PurposeName Purpose# [Abbreviation] )
    ;User-Defined Purposes:
    ;System-Reserved Purposes:
    ( warning 234 wng )
    ( too1 235 tl1 )
    .
    .
    ( cell 254 cel )
) ;end of techPurposes

techLayerPurposePriorities(
    ;layers are ordered from lowest to highest priority
    ;( layerName Purpose )
    ( background drawing )
    ( grid drawing )
    .
    .
    ( Unrouted drawing9 )
) ;end of techLayerPurposePriorities

Layer Definitions class enclosure. All subclasses must be specified within the parentheses of the class enclosure.

Technology Layers subclass. Defines layers. Specifies name, number, and optional abbreviation for each layer. Data for each layer must be enclosed in parentheses. All layer specifications must be enclosed within the parentheses of the subclass enclosure.

Technology Purposes subclass. Defines purposes. Specifies name, number, and optional abbreviation for each purpose. Data for each purpose must be enclosed in parentheses. All purpose specifications must be enclosed within the parentheses of the subclass enclosure.

Technology Layer-Purpose Pair Priorities subclass. Lists all layer-purpose pairs in display priority order, from lowest to highest. Each layer-purpose pair must be enclosed in parentheses. All layer-purpose pairs must be enclosed within the parentheses of the subclass enclosure.
**Technology Displays subclass.** Defines how layers are displayed. Specifies layer name and purpose for each layer-purpose pair and assigns display packet by name. Also specifies whether the layer-purpose pair is visible (displayed in the cellview), selectable (in the cellview), included in a Diva® changed layer, draggable (you can drag a shape created with the layer-purpose pair in the layout editor), and valid (displayed in the LSW). Data for each layer must be enclosed in parentheses. All layer display specifications must be enclosed within the parentheses of the subclass enclosure.

```plaintext
techDisplays(
  ;( layerName  Purpose  Packet  Vis  Sel  chgLay  DrgEnbl  Valid )
  ( background  drawing  background  t  nil  t  nil  nil )
  ( grid  drawing  grid  t  nil  t  nil  nil )
  ( Unrouted drawing9 Unrouted9  t  t  t  t  nil )
) ;end of techDisplays
```

**Technology Layer Properties subclass.** Specifies user-defined properties for layer-purpose pairs. The data for each layer-purpose pair must be enclosed in parentheses. All user-defined property assignments must be enclosed within the parentheses of the subclass enclosure.

```plaintext
techLayerProperties(  
  ;( PropName  Layer1  [Layer2] PropValue )
  ( defaultWidth ndiff  1.000000 )
  ( defaultWidth pdiff  1.000000 )
) ;end of techLayerProperties
```

End of Layer Definitions class enclosure. All subclasses must be enclosed within the parentheses of the class enclosure.
Specifying Layer Definitions

Technology file layer definitions are required. As illustrated in the sample, the class must be specified by enclosing the subclass definitions within the `layerDefinitions()` class enclosure. The following paragraphs provide detailed information about specifying subclass data.

Defining Layers: techLayers()

Layers represent the type of manufacturing material you want to use in your design (for example: `metall`, `poly`, `ndiff`). DFII also uses special layers to display warnings and for highlighting. You need to create the layers that are specific to your designs.

DFII provides system-reserved layers that you can use in your designs. If you need a layer that Cadence has not provided, you can define your own. For a list of the system-reserved layers, refer to Appendix B, “System-Reserved Layers and Purposes.”

Each layer must have a name and a number. You use the layer name to refer to the layer. Layer numbers are used by the Cadence software internally. For layers you define, assign layer numbers 0 through 127. Layer numbers 128 through 255 are for system-reserved layers.

The `techLayers()` subclass (section) of the Layer Definitions class defines the layers to be used in your design.
Technology File and Display Resource File User Guide
Creating a Technology File: Control, Layer, and Device Definitions

**Technology Layers subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Layer name.** Each layer name must be unique. The name represents the type of manufacturing material to use in your design.
- **Layer number.** Specify a unique layer number between 0 and 127. (Layer numbers 128 through 255 are set aside for system-reserved layers.)
- **Layer name abbreviation.** Abbreviations can be up to seven characters long (see Note).

```

```

**Note:** Applications that display layer names do not always have room to display the entire name. The optional abbreviation expands your control over what is displayed in narrow fields. Depending on the width of the field for displaying the layer name, an application displays whichever of the following fits:

- The full layer name
- The layer name truncated to fit (if no abbreviation is specified)
- The abbreviation
- The abbreviation truncated to fit

Hierarchical designs use the technology file of the parent cell to display layers. If you use multiple technology files in your designs, make sure the names and numbers of the layers you need to display are consistent in all technology files. If layer definitions for the child cell conflict with or are missing from the technology file of the parent cell, your design will contain errors. Refer to Appendix C, “Resolving Layer Errors,” for information on how to fix the layer problems.

For more information about the `techLayers()` section, refer to the Technology File and Display Resource File SKILL Reference Manual.
Defining Layer Purposes: techPurposes()

The layer purpose indicates how you use a layer in your design. For example, you could use the purposes drawing, pin, and net to distinguish the various uses of the layer metal1 in a design. By creating layer-purpose pairs using the same layer name and different purposes, you can use the same layer in several ways in a design. For example, to distinguish between the polygon and pin data on the metal1 layer, you can use the metal1 drawing and metal1 pin layer-purpose pairs.

DFII provides system-reserved purposes that satisfy most of your design needs. If you need a purpose that Cadence has not provided, you can define your own. For a list of the system-reserved purposes, refer to Appendix B. “System-Reserved Layers and Purposes.”

A purpose has a name and a number. You use the purpose name to refer to the purpose. Purpose numbers are used internally by the Cadence software. For purposes you define, use purpose numbers 1 through 127. Purpose numbers 128 through 255 are for system-reserved layer purposes.

The techPurposes subclass (section) of the Layer Definitions class defines the purposes to be used in your design.

**Technology Purposes subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

```
| Purpose name. Each purpose name must be unique. The name represents the layer purpose. |
| Purpose number. Specify a unique purpose number between 1 and 127. (Purpose numbers 128 through 255 are for system-reserved purposes.) |
| Purpose name abbreviation. Abbreviations can be up to seven characters long (see Note). |
```

```
techPurposes(
    ; ( PurposeName   Purpose#   [Abbreviation] )
    ( power 1 pwr )
    ( ground 2 gnd )
); end of techPurposes
```

**Note:** Applications that display purpose names do not always have room to display the entire name. The optional abbreviation expands your control over what is displayed in narrow fields. Depending on the width of the field for displaying the purpose name, an application displays whichever of the following fits:
The full purpose name

The abbreviation

The first and last letters of the full purpose name

For more information about the techPurposes() section, refer to the Technology File and Display Resource File SKILL Reference Manual.

Defining Layer Display Priorities: techLayerPurposePriorities

To display your design properly, you need to control the order in which applications display the layers. For example, it is important to specify which routing layers appear on top of other routing layers.

The techLayerPurposePriorities subclass (section) of the Layer Definitions class defines the display priority order of layer-purpose pairs. In this subclass, you list the layer-purpose pairs in the order in which they are to be displayed; applications display a layer-purpose pair on top of the one that precedes it in the list and underneath the one that follows it in the list.

Technology Layer-Purpose Pair Priorities subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

Layer-Purpose Pair (Layer name. Purpose name.). Specify the layer-purpose pairs in the order in which you want them displayed. Layer-purpose pairs listed last are displayed on top of the layer-purpose pairs listed first. Each layer-purpose pair must be unique and must be enclosed in parentheses.

```
techLayerPurposePriorities(
  ; ( layerName Purpose )
  ( nwell drawing )
  ( nwell net )
  ( nwell pin )
  ( pwell drawing )
  ( pwell net )
  ( pwell pin )
)
```

For more information about the techLayerPurposePriorities() section, refer to the Technology File and Display Resource File SKILL Reference Manual.
Assigning Display Packets and Defining Layer Display Attributes: techDisplays()

Display packets, which are defined in the display resource file and assigned to layers in the technology file, control how layers appear on your monitor and plotting devices. Display attributes control

- Whether the layer is visible in the cellview
- Whether the layer is selectable in the cellview
- Whether the layer is included in the Diva change layer
- Whether an object is visible when dragged (displayed when you move it)
- Whether the layer is displayed in the layer selection window (LSW)

The `techDisplays` subclass (section) of the Layer Definitions class assigns display packets to layer-purpose pairs and specifies layer display attributes.
**Technology Displays subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

**Layer name. Purpose name.** Specify the layer-purpose pair. The order of the layer-purpose pairs is not important.

**Display Packet Name.** List a packet that is defined in the display resource file (display.drf).

**Visible.** When set to t, objects are visible.

**Selectable.** When set to t, objects are selectable.

**Changed layer.** When set to t, the Diva software tracks changes to objects in incremental verification.

**Drag.** When set to t, an object is displayed as it moves.

**Valid.** When set to t, the layer-purpose pair is displayed on the LSW.

```
techDisplays(
  ; ( Layer  Purpose  Packet  Vis  Sel  ChgLay  Drag  Valid )
  ( nwell  drawing  yelhash_S  t  t  t  t  t )
  ( nwell  net    yelhash_S  t  t  nil nil nil )
  ( nwell  pin    yelhash_S  t  nil t  t  t )
  ( pwell  drawing grnhash_S  t  t  nil t  t )
  ( pwell  net    grnhash_S  nil t  t  t  t )
  ( pwell  pin    grnhash_S  t  t  t  t  t )
) ; end techDisplays
```

For more information about display packets, refer to Chapter 6, “Creating a Display Resource File.” For more information about the techDisplays() section, refer to the Technology File and Display Resource File SKILL Reference Manual.
Specifying User-Defined Layer Properties: techLayerProperties()

The techLayerProperties subclass (section) of the Layer Definitions class assigns user-defined properties to layer-purpose pairs.

**Technology Layer-Purpose Pair Properties subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Property name.** Specify a property name.
- **Layer 1.** Specify the layer name or layer-purpose pair (enclosed in parentheses) for a layer to which to apply the property.
- **Layer 2.** Optionally specify the layer name or layer-purpose pair (enclosed in parentheses) for a second layer to which to apply the property.
- **Property value.** Specify a property value.

```java
techLayerProperties{
    ; ( PropName Layer1 [Layer2] PropValue )
    ( defaultWidth ndiff 1.000000 )
    ( defaultWidth (pdiff drawing) 1.000000 )
}; end techLayerProperties
```

For more information about the techLayerProperties() section, refer to the Technology File and Display Resource File SKILL Reference Manual.

**Devices**

Layout is a method of creating custom integrated circuit layouts using devices such as transistors, contacts, and pins instead of polygons.

Technology file devices are similar to parameterized cells except that these devices can be defined only in the technology file. In addition, technology file devices have two levels of parameterization, **class parameters** and **formal parameters**. Class parameters control values for the physical construction of a device, such as the diffusion layer or gate layer. Formal parameters define values you can modify when you place an instance of a device; for example, the width of a transistor or the spacing of contacts in an array. The formal parameters appear in the Create Device form when you place instances of a device.
Cadence provides a set of predefined device types; you can also create your own device types. The following are the Cadence-predefined device types:

- **contact**
  - Single contact or rectangular array of contacts placed between two layers
- **ruleContact**
  - Single contact or rectangular array of contacts placed between multiple (three or more) layers
  
  **Note**: Used only by Preview Silicon Ensemble™ and Preview Gate Ensemble® place-and-route software to define special routing vias.
- **enhancement**
  - MOS transistor with overlapping dot pins in its center for the gate, source, and drain
- **depletion**
  - MOS transistor with overlapping dot pins in its center for the gate, source, and drain, plus a depletion layer over the channel region
- **pin**
  - Square dot pin; instance, single- or double-layer
- **rectPin**
  - Rectangular dot pin

The definitions for the predefined device types are in the sample technology file

```text
install_dir/samples/techfile/devices.tf
```

LAS uses all of the predefined devices plus two additional device types, `syMGEnhancement` and `syMGDepletion`. These device definitions are defined in the sample technology file

```text
install_dir/samples/techfile/multGate.tf
```

To establish a device as part of your technology data, you must specify two sets of data:

- **Device-type definition**
  - **Class parameters**
  - **Formal parameters**
  - **SKILL code describing the physical geometries of the device**

Every device type is defined by a set of class and formal parameters. Predefined device types have fixed sets of formal and class parameters. You do not need to define these device types, but you must specify a statement to activate them. When you create a custom device type, you create your own parameters as you specify your own device type definition.
Device declaration

When you declare an individual device, you fill in specific values for the parameters that make up the device type definition. You must declare each device you want to use in your design.

You can also specify properties for device types and individual devices.

Several of the DFII applications use the devices defined in the technology file when processing a design. For example:

- the Virtuoso Layout Synthesizer (LAS), Virtuoso XL, Preview Silicon Ensemble place-and-route software, and Preview Gate Ensemble place-and-route software place devices when they generate a layout.
- the Virtuoso compactor uses information stored with the device when it compacts a design.

This section describes how to define devices in the technology file. For information about the device data the various DFII applications use, refer to the following:

- “Setting up the Technology File for LAS” in the Virtuoso Layout Synthesizer User Guide
- “Getting Started” in the Virtuoso Compactor Reference Manual
- “LEF and DEF Mapping” in the Preview Silicon Ensemble Reference
- “LEF Data Map” in the Preview Gate Ensemble Reference

Sample Devices Class

The following sample Devices class illustrates the class and its subclasses (sections), along with the devices they define and declare.

For more information about the Devices class, refer to the Technology File and Display Resource File SKILL Reference Manual.
Creating a Technology File: Control, Layer, and Device Definitions

devices(
    tcCreateCDSDeviceClass()
)

symContactDevice(
    ;( deviceName viaLayer viaPurpose
    ( M1_P cont drawing
    ; layer1 purpose1 [implant1]
    diff drawing (pimplant drawing 0.3)
    ; layer2 purpose2 [implant2]
    metall drawing
    ; width length [( row column xPitch yPitch xBias yBias )]
    0.6 0.6 ( 1 1 _NA_ _NA_ _NA_ _NA_ )
    ; encLayer1 encLayer2 legalRegion
    0.6 0.6 ( outside nwell drawing )
    )
); end of symContactDevice

ruleContactDevice(
    ;( deviceName
    ( "VIABIGPOWER12"
    ;( layer1 purpose1 rectangles ) | nil
    ( diff drawing (-21.000 -21.000 21.000 21.000 )
    ;[( viaLayer viaPurpose rectangles )]
    ( cont drawing (-2.400 -0.800 2.400 0.800 )
    (-19.000 -17.400 -14.200 19.000 )
    ( 14.200 -19.000 19.000 -17.400 )
    (-19.000 -0.800 -14.200 0.800 )
    ( -2.400 -19.000 2.400 -17.400 )
    ( 14.200 -0.800 19.000 0.800 )
    )
    ;[( layer2 purpose2 rectangles )]
    ( metall drawing (-21.000 -21.000 21.000 21.000 )
    .
    ); end of ruleContactDevice

Devices class enclosure. All subclasses must be specified within the parentheses of the class enclosure. The Devices class usually follows the Layer Definitions class.

Create Cadence Device Type subclass. Activates all of the Cadence-predefined device types.

Declare Contact Device subclass. Declares contact devices. Data for each contact device must be enclosed in parentheses. All contact device specifications must be enclosed within the parentheses of the subclass enclosure.

Declare Rule Contact Device subclass. Declares rule contact devices. Data for each contact device must be enclosed in parentheses. All contact device specifications must be enclosed within the parentheses of the subclass enclosure.
symEnhancementDevice(
    ;( deviceName sdLayer sdPurpose
    ( PTR diff drawing
    ; [ implantEnclosure ]
    ( pimplant drawing 0.3 )
    ; gateLayer gatePurpose
    polyl  drawing
    ; width length sdExt gateExt legalRegion  
    1.8 0.6 1.2 0.9 ( outside pwell drawing )
    )
    ;( deviceName sdLayer sdPurpose
    ( NTR diff drawing
    ; [ implantEnclosure ]
    ; gateLayer gatePurpose
    polyl  drawing
    ; width length sdExt gateExt legalRegion
    1.8 0.6 1.2 0.9 ( inside pwell drawing )
    )
) ; end of symEnhancementDevice

symDepletionDevice(
    ;( deviceName sdLayer sdPurpose
    ( D_NTR diff drawing
    ; [ implantEnclosure ]
    ; gateLayer gatePurpose deplLayer deplPurpose
    polyl drawing depletion drawing
    ; width length sdExt gateExt deplEncSD deplEncGate
    1.8 0.6 1.2 0.9 0.3 0.3
    ; [ legalRegion ]
    ( outside pwell drawing )
    )
) ; end of symDepletionDevice

Declare Enhancement Device subclass. Declares enhancement devices. Data for each enhancement device must be enclosed in parentheses. All enhancement device specifications must be enclosed within the parentheses of the subclass enclosure.

Declare Depletion Device subclass. Declares depletion devices. Data for each depletion device must be enclosed in parentheses. All depletion device specifications must be enclosed within the parentheses of the subclass enclosure.
Declare Pin Device subclass. Declares square dot pin devices. Data for each pin device must be enclosed in parentheses. All pin device specifications must be enclosed within the parentheses of the subclass enclosure.

Declare Rectangular Pin Device subclass. Declares rectangular dot pin devices. Data for each rectangular dot pin device must be enclosed in parentheses. All rectangular dot pin device specifications must be enclosed within the parentheses of the subclass enclosure.
tcCreateDeviceClass:

; viewName className
"symbolic" "syMGEnhancement"
;
; class parameters
( (paramName value ) (paramName value ) ... )
( (sdLayer "hilite") (gateLayer "hilite")
(sdExt 0.0) (gateExt 0.0 )
(sdImpLayer nil) (sdImpEnc 0.0 ) ) ;
; formal parameters
( (paramName value ) (paramName value ) ... )
( (width 0.0 ) (length 0.0 ) )
; geometry
W2 = width/2 L2 = length/2
netId = dbMakeNet(tcCellView "G")
dbId = dbCreateDot(tcCellView gateLayer -W2-gateExt:0)
dbId = dbCreatePin(netId dbId "gl")
dbSetq(dbId list("left") accessDir)
dbId = dbCreatePin(netId dbId "gr")
dbSetq(dbId list("right") accessDir)
dbId = dbCreateRect(tcCellView gateLayer
list(-W2-gateExt:-L2 W2+gateExt:L2))
dbAddFigToNet(dbId netId)
;
netId = dbMakeNet(tcCellView "S")
dbId = dbCreateDot(tcCellView sdLayer 0:L2)
dbId = dbCreatePin(netId dbId "s")
dbSetq(dbId list("top") accessDir)
dbId = dbCreateRect(tcCellView sdLayer list(-W2:0 W2:L2+sdExt) )
dbAddFigToNet(dbId netId)
);
) ;end of tcCreateDeviceClass

tcDeclareDevice:

; viewName className deviceName
"symbolic" "syMGEnhancement" "MGPTR"
;
; class parameters
( (paramName value ) (paramName value ) ... )
( (gateLayer "poly1") (sdLayer "diff")
(sdExt 1.200000) (gateExt 0.300000 ) ) ;
; formal parameters
( (paramName value ) (paramName value ) ... )
( (w 1.8 ) (l 0.6 ) )
);
; end of tcDeclareDevice

) ;end of devices class

Create Custom Device Class subclass. Creates and defines a custom (user-defined) device type. You must specify a separate statement for each device type you define.

Declare Device subclass. Declares a device. The device type can be either Cadence-predefined or user-defined. You must specify a separate statement for each device you declare.

End of Devices class enclosure. All subclasses must be enclosed within the parentheses of the class enclosure.
Specifying Devices

Using technology file devices is optional. If you do want to use devices, however, technology file device specifications are required. The Devices class contains information about devices used by DFII applications to generate or optimize your designs. As illustrated in the sample Devices class, the class must be specified by enclosing the subclass definitions within the `devices()` class enclosure. The following paragraphs provide detailed information about specifying Devices class and subclass data.

Specifying Predefined Device Types

To use the Cadence device types, you must first activate all of them with `tcCreateCDSDeviceClass`. After activating the predefined device types, you declare devices of specific predefined types with their corresponding technology file subclasses — or, alternatively, you can specify the more general `tcDeclareDevice()` subclass to declare a predefined device type.

Cadence provides six predefined device types. This section shows you the statements you must place in the Devices class to initialize these device types so you can use them to create devices.

Activating Cadence-Predefined Device Types: `tcCreateCDSDeviceClass()`

The `tcCreateCDSDeviceClass()` subclass (section) activates all of the Cadence-predefined device types. This subclass takes no parameters. It must, however, appear before declarations for devices of the Cadence-predefined types.

For more information about the `tcCreateCDSDeviceClass()` section, refer to the Technology File and Display Resource File SKILL Reference Manual.

Declaring a Contact Device between Two Layers: `symContactDevice()`

A contact device is a single contact or rectangular array of contacts placed between two layers. The layers overlap the contacts uniformly and a dot pin is placed in the center of each contact.
The `symContactDevice()` subclass (section) of the Devices class declares contact devices. The following example defines a single contact named M1_P:

```plaintext
symContactDevice(
  ( deviceName viaLayer viaPurpose layer1 purpose1 
    ( M1_P cont drawing diff drawing (pImplant drawing 0.3) 
    ; layer2 purpose2 width length 
      [implant1] (pimplant drawing 0.3) 
    )
  ; encLayer1 encLayer2 legalRegion )
) ; end of symContactDevice
```

The following illustration shows what the contact looks like:

The `xPitch` and `yPitch` values control spacing for contact arrays. The `xPitch` value defines the minimum allowable space, origin-to-origin (center-to-center), between consecutive contacts in a horizontal row of the array, while `yPitch` controls the minimum allowable spacing between contact origins (that is, center-to-center) in vertical rows. If `xPitch` and `yPitch` are not specified, the default pitches are as follows:

- **xPitch**: Minimum allowable space, origin-to-origin (center-to-center), between consecutive contacts in a horizontal row of the array.
- **yPitch**: Minimum allowable spacing between contact origins (that is, center-to-center) in vertical rows.

The default pitches are set to 0.6 for both directions.
xPitch is the sum of the contact width plus the minimum spacing value for the via layer. If no minimum spacing is defined for the via layer, the software adds the contact width to the smaller of the enclByLayer1 or enclByLayer2 values.

yPitch is calculated like xPitch, using the length value instead of the width value.

The xBias and yBias values determine where the array origin is placed. Possible values are calculated as follows:

- xBias can be left, center, or right. The default is center.
- yBias can be top, center, or bottom. The default is center.

Adding the following information changes the sample contact to a contact array:

```
; ( row column xPitch yPitch xBias yBias )
( 2 2 1.2 1.2 center center )
```

The contact array appears as follows:

```
xPitch 1.2
```

```
Origin
```

```
yPitch 1.2
```

Declaring a Contact Device between Multiple Layers: ruleContactDevice()

A rule contact device is a single contact or rectangular array of contacts placed between multiple (three or more) layers. Rule contact devices are used by Preview Silicon Ensemble and Preview Gate Ensemble place-and-route software to define special routing vias. These vias are used in conjunction with the viaRule, genViaRule, and nondefaultRule sections of the technology file.

The ruleContactDevice() subclass (section) of the Devices class declares rule contact devices. The following example defines a rule contact named VIA12:
Declaring an Enhancement Device: symEnhancementDevice()

An enhancement device is a MOS transistor. In the center of the transistor are overlapping dot pins for the gate, source, and drain.

The symEnhancementDevice() subclass (section) of the Devices class declares enhancement devices. The following examples define a PTR device and an NTR device.
Creating a Technology File: Control, Layer, and Device Definitions

The devices appear as follows:

```
symEnhancementDevice(
  ;( deviceName sdLayer sdPurpose [ implantEnclosure ]
  ( PTR diff drawing ( pimplant drawing 0.3 )
    ; gateLayer gatePurpose polyl drawing
      gate layer and purpose
    ; width length sdExt gateExt legalRegion )
    1.8 0.6 1.2 0.9 ( outside pwell drawing )
    )
);

; ( deviceName sdLayer sdPurpose [ implantEnclosure ]
( NTR diff drawing
  ; gateLayer gatePurpose polyl drawing
    ; width length sdExt gateExt legalRegion )
    1.8 0.6 1.2 0.9 ( inside pwell drawing )
    )
);

; ) end of symEnhancementDevice
```

The devices appear as follows:

- **NTR device**
  - Length 0.6
  - Gate extension 0.9
  - Source/drain extension 1.2
  - Width 1.8

- **PTR device**
  - Implant layer spacing is 0.3
Declaring a Depletion Device: symDepletionDevice()

A depletion device is an enhancement device with a depletion layer over the channel region.

The `symDepletionDevice()` subclass (section) of the Devices class declares depletion devices. The following example defines a depletion device called D_NTR. (Note: This example does not include the optional implant enclosure, although the syntax allows for one. For an example, refer to the enhancement device in the previous section.)

```plaintext
symDepletionDevice{
  ; ( deviceName  sdLayer  sdPurpose  [ implantEnclosure ]
   D_NTR  diff  drawing
   device name  source/drain layer and purpose

   ; gateLayer  gatePurpose  deplLayer  deplPurpose
   poly1  drawing  depletion  drawing

   gate layer and purpose  depletion layer and purpose

   ; width  length  sdExt  gateExt  deplEncSD  deplEncGate
   1.8  0.6  1.2  0.9  0.3  0.3

   gate extension  spacing of depletion enclosure around gate
   length  spacing of depletion enclosure around source/drain

   ; [ legalRegion ] )
   ( outside pwell drawing )

) ; end of symDepletionDevice
```

The device appears as follows:
Declaring a Square Dot Pin Device: symPinDevice()

A pin device is a square dot pin. Pin devices can be single-layer or double-layer devices.

**Note:** Applications that automatically place pins require that the layers used to draw the pin have a purpose of drawing.

The `symPinDevice()` subclass (section) of the Devices class declares pin devices. The following example is a single-layer pin device named `poly1_T`.

```plaintext
symPinDevice(
    ; ( deviceName maskable layer1 purpose1 width1
    ( poly1_T nil poly1 drawing 0.6
      
      pin name
      specifies whether pin is maskable
      pin layer1
      layer1 width
      
      ; [ layer2 purpose2 width2 ] [ legalRegion ]
      [ _NA_ _NA_ _NA_ _NA_ ]

      pin layer2
      layer2 width
      specifies whether pins must be inside or outside the listed well layer
    )

  ) ; end of symPinDevice

) ; end of symPinDevice
```

The pin appears as follows:

```
<table>
<thead>
<tr>
<th>Width 0.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>poly1 layer</td>
</tr>
</tbody>
</table>
```

Declaring a Rectangular Dot Pin Device: symRectPinDevice()

A rectangular pin device is a rectangular dot pin.
The symRectPinDevice() subclass (section) of the Devices class declares rectangular pin devices. The following example declares a simple poly rectangular pin named nplus_P:

```
symRectPinDevice(
  nplus_P nil poly1 drawing 1 2 _NA_ )

) ; end of symRectPinDevice
```

The rectangular pin looks as follows:

```
poly1 layer      Width 1
                Length 2
```

### Specifying Custom, User-Defined Device Types

In addition to using the predefined device type definitions, you can create your own device types. This section shows you the statements you must place in the Devices class to specify customized device data in your technology file. You can create your own device types with `tcCreateDeviceClass` in the technology file and then declare devices of the types you have defined with `tcDeclareDevice`. 
Creating User-Defined Device Types: \texttt{tcCreateDeviceClass()}

The \texttt{tcCreateDeviceClass()} subclass (section) of the Devices class defines a custom device type for use in your design.

Create Custom Device Type subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

- **View name.** Specify the name of the view for devices of this type to have.
- **Class name.** Specify the name of the device type to define.

\begin{verbatim}
\texttt{tcCreateDeviceClass(}
    ; viewName      className
    "symbolic"     "syMGEnhancement"

    ; class parameters        formal parameters
    ; ((paramName value)...   ((paramName value)...)
        ((p1  0.0) (p2  1.0))   ((fp1 0.0) (fp2  0.0))

    ; geometry

    : ; See sample geometry section
\texttt{)} ; end of \texttt{tcCreateDeviceClass} and syMGEnhancement device type definition
\end{verbatim}
**Class parameters.** Class parameters are fixed once you declare a device. They usually define the physical characteristics of the device. Specify the parameter name followed by a default value for each parameter, enclosed in parentheses and separated by a space. You can specify any number of parameters and assign them any names you want. Default values identify the type of value you need to provide when you declare a device. The entire parameter specification section must be enclosed in parentheses.

**Formal parameters.** Formal parameters appear on the options form when you place an instance of a declared device of this type in a layout. Specify the parameter name followed by a default value for each parameter, enclosed in parentheses and separated by a space. You can specify any number of parameters and assign them any names you want. Default values identify the type of value you need to provide when you declare a device. The entire parameter specification section must be enclosed in parentheses.

```
tcCreateDeviceClass(
  ; viewName  className
  "symbolic"  "syMGEnhancement"
  ; class parameters  formal parameters
  ; ((paramName value)...)  ((paramName value)...)  
  ((p1 0.0) (p2 1.0))  ((fp1 0.0) (fp2 0.0))
  ; geometry
  ;  ; See the sample geometry section below
); end of tcCreateDeviceClass and syMGEnhancement device type definition
```

**Note:** Continued in following figure.
**Geometry.** The geometry is a listing of SKILL functions that specify how the class and formal parameters are used to create the device. The geometry you specify builds the device from the parameters identified. You use the SKILL database access commands (db commands) to define the rectangles, lines, nets, terminals, and pins that make up the device type.

```skill
tcCreateDeviceClass(
    ; viewName className
    "symbolic" "syMGEnhancement"
    ; class parameters formal parameters
    ; ((paramName value)...) ((paramName value)...) ((p1 0.0) (p2 1.0)) ((fp1 0.0) (fp2 0.0))
    ; geometry
    .
    . ; See the sample geometry section below
);
); end of tcCreateDeviceClass and syMGEnhancement device type definition
The following is a sample geometry section:

```
; geometry
  W2 = width/2   L2 = length/2
  netId = dbMakeNet(tcCellView "G")
  dbId = dbCreateDot(tcCellView gateLayer -W2-gateExt:0)
  dbId = dbCreatePin(netId dbId "gl")
  dbSetq(dbId list("left") accessDir)
  dbId = dbCreateDot(tcCellView gateLayer W2+gateExt:0)
  dbId = dbCreatePin(netId dbId "gr")
  dbSetq(dbId list("right") accessDir)
  dbId = dbCreateRect(tcCellView gateLayer list(-W2-gateExt:-L2 W2+gateExt:L2))
  dbAddFigToNet(dbId netId)
;
  netId = dbMakeNet(tcCellView "S")
  dbId = dbCreateDot(tcCellView sdLayer 0:L2)
  dbId = dbCreatePin(netId dbId "s")
  dbSetq(dbId list("top") accessDir)
  dbId = dbCreateRect(tcCellView sdLayer list(-W2:0 W2:L2+sdExt))
  dbAddFigToNet(dbId netId)
;
  netId = dbMakeNet(tcCellView "D")
  dbId = dbCreateDot(tcCellView sdLayer 0:-L2)
  dbId = dbCreatePin(netId dbId "d")
  dbSetq(dbId list("bottom") accessDir)
  dbId = dbCreateRect(tcCellView sdLayer list(-W2:-L2-sdExt W2:0))
  dbAddFigToNet(dbId netId)
;
  if( sdImpLayer then
    dbCreateRect(tcCellView sdImpLayer
                  list(-W2-sdImpEnc:-L2-sdExt-sdImpEnc
                        W2+sdImpEnc:L2+sdExt+sdImpEnc))
```

For more information about the `tCreateDeviceClass()` section, refer to the *Technology File and Display Resource File SKILL Reference Manual*.

### Declaring User-Defined Devices: `tcDeclareDevice()`

The `tcDeclareDevice()` subclass (section) of the Devices class declares a device of whatever type you specify for use in your design. You must declare customer (user-defined) devices with `tcDeclareDevice()`; you can also declare devices of Cadence-predefined types rather than declaring them with the device-type specific statements if you wish.
Declare Device subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure. This subclass must come after the section (tcCreateCDSDeviceClass() or tcCreateDeviceClass()) that defines the device type. Specify a separate tcDeclareDevice() statement for each device you declare.

View name. Specify the type of view for this device. This is the view of the cellview that is generated when you compile the technology file. It must match the view defined for the device type.

Class name. Specify the device class name. It must be a defined or predefined device type.

Device name. Specify the name of the device to declare.

tcDeclareDevice(
; viewName className deviceName
"symbolic" "syMGEnhancement" "MGPTR"

; class parameters formal parameters
; ((paramName value)...) ((paramName value) ... )
(p1 0.0) (p2 1.0) (fp1 0.0) (fp2 0.0)
)
; end of tcDeclareDevice and syMGEnhancement device declaration

Class parameters. Specify the parameter name defined in the device type definition followed by the value you want to specify for each parameter, enclosed in parentheses and separated by a space. The value must be of the same type as the default parameter value defined for the device type. The entire parameter specification section must be enclosed in parentheses.

Formal parameters. Specify the parameter name defined in the device type definition followed by the value you want to specify for each parameter, enclosed in parentheses and separated by a space. The value must be of the same type as the default parameter value defined for the device type. The entire parameter specification section must be enclosed in parentheses.

tcDeclareDevice(
; viewName className deviceName
"symbolic" "syMGEnhancement" "MGPTR"

; class parameters formal parameters
; ((paramName value)...) ((paramName value) ... )
(p1 0.0) (p2 1.0) (fp1 0.0) (fp2 0.0)
)
; end of tcDeclareDevice and syMGEnhancement device declaration

For more information about the tcDeclareDevice() section, refer to the Technology File and Display Resource File SKILL Reference Manual.
Applying Properties to Devices

Some applications require you to apply properties to a device type so it can be placed properly in a generated design. You can also apply user-defined properties to a device type or an individual device.

Applying Properties to a Device Type: tcSetDeviceClassProp()

The \texttt{tcSetDeviceClassProp()} subclass (section) of the Devices class applies properties to a device type.

\textbf{Declare Device Class Properties subclass enclosure.} All subclass data must be specified within the parentheses of the subclass enclosure. This subclass comes immediately after the section (\texttt{tcCreateCDSDeviceClass()} or \texttt{tcCreateDeviceClass()} that defines the device type.

\begin{itemize}
  \item \textbf{View type.} Specify the type of view for this device type. This is the view of the cellview that is generated when you compile the technology file. It must match the view defined for the device type.
  \item \textbf{Device type.} Specify the name of the device type to which to apply properties.
  \item \textbf{Property and value.} Specify the name of the property to apply followed by the value you want to specify for the property, enclosed in parentheses and separated by a space.
\end{itemize}

\begin{verbatim}
tcSetDeviceClassProp(
  ; view deviceType (property value )
  "symbolic" "syMGDepletion" ("function" "transistor" )
  .
  .
) ; end of tcSetDeviceClassProp
\end{verbatim}
Applying Properties to an Individual Device: tcSetDeviceProp()

The tcSetDeviceProp() subclass (section) of the Devices class applies properties to an individual device.

Declare Device Properties subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure. This subclass comes immediately after the section that declares the device.

**View type.** Specify the type of view for this device. This is the view of the cellview that is generated when you compile the technology file. It must match the view defined for the device type.

**Device name.** Specify the name of the device to which to apply properties.

**Property and value.** Specify the name of the property to apply followed by the value you want to specify for the property, enclosed in parentheses and separated by a space. The entire parameter specification section must be enclosed in parentheses.

```
tcSetDeviceProp(
    ; view deviceName (property value )
    "symbolic" "MGPTR" ("legalRegion" (outside pwell drawing))
    ...
    ); end of tcSetDeviceProp
```
Creating a Technology File: Generic Rules

This chapter discusses the technology file classes that specify generic rules, which apply to all applications and specify the following:

- “Layer Rules” on page 78
- “Physical Rules” on page 83
- “Electrical Rules” on page 90

Other technology file classes define the following:

- Controls, described in Chapter 3, “Creating a Technology File: Control, Layer, and Device Definitions.”
- Layer definitions, described in Chapter 3, “Creating a Technology File: Control, Layer, and Device Definitions.”
- Devices, described in Chapter 3, “Creating a Technology File: Control, Layer, and Device Definitions.”
- Application-specific rules, described in Chapter 5, “Creating a Technology File: Application-Specific Rules.”

Layer Rules

You specify layer rules to establish the relationships and interactions between layers. Layer rules define the following:

- Via layers that connect two conducting layers
- Layers that are physically and electrically equivalent
- Stream translation data for a layer

Layer rules are specified in the Layer Rules class, a generic rules class that can be used by all applications.
Sample Layer Rules Class

The following sample Layer Rules class illustrates the class and its subclasses (sections), along with the layer rules they define.


```plaintext
layerRules(

viaLayers(
    ; ( layer1 viaLayer layer2 )
    ( poly cont (metal1 net))
    ( (metal1 net) via (metal2 net))
    ; end of viaLayers

equivalentLayers(
    ; ( (layer)...) (metal1 net) (Vdd net) (Gnd net) )
    ;end of equivalentLayers

streamLayers(
    ; ( layer streamNum dataType translate )
    ( ndiff 1 0 t )
    ( pdiff 2 0 t )
    ;end streamLayers
)
)
```

Layer Rules class enclosure. Follows Devices class. All subclasses must be specified within the parentheses of the class enclosure.

Via Layers subclass. Defines layers that conduct between two other layers. Specifies bottom routing layer, middle (via) layer, and top routing layer. Data for each via layer must be enclosed in parentheses. All layer specifications must be enclosed within the parentheses of the subclass enclosure.

Equivalent Layers subclass. Lists layer-purpose pairs that represent the same type of material. Each layer-purpose pair must be enclosed in parentheses, and layer-purpose pairs that are equivalent must be within parentheses. All equivalent layers specifications must be enclosed within the parentheses of the subclass enclosure.

Stream Layers subclass. Lists stream translation data for the layer-purpose pairs to translate using pipo. The data for each layer-purpose pair must be enclosed in parentheses. All stream translation data must be enclosed within the parentheses of the subclass enclosure.

End of Layer Rules class enclosure. All subclasses must be enclosed within the parentheses of the class enclosure.
Specifying Layer Rules

Technology file layer rules are required. As illustrated in the sample, the class must be specified by enclosing the subclass definitions within the `layerRules()` class enclosure. The following paragraphs provide detailed information about specifying subclass data.

For more information about the `layerRules` class, refer to the *Technology File and Display Resource File SKILL Reference Manual*.

Defining Via Layers: `viaLayers()`

A via layer is a layer that connects two conducting layers. In the `layerRules` class, you specify the bottom routing layer, the middle (via) layer, and the top routing layer; you specify further via layer data in the rules class for the application you are using. The following applications use via layers data specified in the technology file:

- Virtuoso® layout editor software
- Virtuoso layout acceleration (Virtuoso XL) software
- Preview Silicon Ensemble™ place-and-route software
- Preview Gate Ensemble® place-and-route software
The `viaLayers` subclass (section) of the Layer Rules class specifies the via layers for your design.

**Via Layers subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

Specify one via layer definition line for each via layer rule. Each line defines the following:

1. **Conducting layer 1.** Specify the layer name or layer-purpose pair (enclosed in parentheses) for the bottom routing layer (the layer closest to the substrate).
2. **Via layer.** Specify the layer name or layer-purpose pair (enclosed in parentheses) for the middle layer (the via layer).
3. **Conducting layer 2.** Specify the layer name or layer-purpose pair (enclosed in parentheses) for the top routing layer (the layer furthest from the substrate).

```
viaLayers(
    ; ( layer1 viaLayer layer2 )
    ( poly1 cont (metal1 net) )
    ( metal1 via metal2 )
    ( metal2 via2 metal3 )
) ; end of viaLayers
```

For more information about the `viaLayers` section, refer to the *Technology File and Display Resource File SKILL Reference Manual.*

**Defining Equivalent Layers: equivalentLayers()**

Equivalent layers are layers that have the same physical and electrical properties or characteristics. You define equivalent layers in the `equivalentLayers` section of the Layer Rules class of the technology file.

The following applications use technology file equivalent layers data:

- Virtuoso XL
- Virtuoso layout synthesizer (LAS)
The equivalentLayers subclass (section) of the Layer Rules class specifies the equivalent layers for your design.

**Equivalent Layers subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

Equivalent layers. Specify the layer name or layer-purpose pair (enclosed in parentheses) for all layers that are equivalent to each other. Each set of equivalent layers must be enclosed within its own set of parentheses.

```plaintext
equivalentLayers(
    ; ( layer...
        ( (metal1 net) (Pwr net) (Pwr1 net) )
        ( (metal1 pin) (Gnd pin) (Gnd1 pin) )
        ( poly poly1 poly2 )
    )
); end of equivalentLayers
```

For more information about the equivalentLayers section, refer to the *Technology File and Display Resource File SKILL Reference Manual*.

**Defining Stream Translation Rules: streamLayers()**

Stream translation rules control file conversion to the Cadence® GDSII Stream format. These rules must be set in the technology file to perform the conversion. For more information about using the Stream translators and GDSII Stream format, refer to the *Design Data Translator’s Reference*. 
The streamLayers subclass (section) of the Layer Rules class specifies the stream layers for your design.

**Stream Layers subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Layer.** Specify the layer name or layer-purpose pair (enclosed in parentheses) for the layer to translate. The data for each layer must be enclosed within its own set of parentheses.
- **Stream number.** Assign a stream number (0–255) to the layer. Default: the layer number.
- **Stream data type.** Specify the stream data type (0–127) for the layer. Default: 0.
- **Translate.** Specify whether to translate the layer (t or nil). Default: t.

```
streamLayers(
    ; ( layer streamNum dataType translate )
    ( (metall net) 1 0 t )
    ( ndiff 2 0 t )
    ( pdiff 3 0 t )
) ; end of streamLayers
```

For more information about the streamLayers section, refer to the *Technology File and Display Resource File SKILL Reference Manual.*

**Physical Rules**

You specify physical rules to establish spacing within and between objects in your design and to specify the grid snapping. Physical rules are specified in the Physical Rules class, a generic rules class that can be used by all applications. Physical rules define the following:

- Spacing information for individual objects; for example, width and notch spacing rules
- Spacing information for two objects; for example, the minimum distance allowed between objects on the same layer or different layers
- The amount of space required when one object encloses another
- The manufacturing grid resolution
Spacing rules specify the distance required between layers and the width of objects and paths. Ordered spacing rules specify the distance required when one layer encloses another.

The following table shows the rules you can set, the applications that use each rule, and the technology file section in which to define each rule:

<table>
<thead>
<tr>
<th>Type of Spacing Rule</th>
<th>Rule</th>
<th>Applications That Use It</th>
<th>Section of Physical Rules Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum width of a path on the specified layer</td>
<td>minWidth</td>
<td>Layout Editor, Virtuoso XL, LAS, Compactor, Preview Silicon Ensemble, Preview Gate Ensemble, Diva</td>
<td>spacingRules</td>
</tr>
<tr>
<td>Default width of a path on the specified layer</td>
<td>defaultWidth</td>
<td>Compactor</td>
<td>spacingRules</td>
</tr>
<tr>
<td>Maximum width of a path on the specified layer</td>
<td>maxWidth</td>
<td>Layout Editor, Virtuoso XL, Compactor</td>
<td>spacingRules</td>
</tr>
<tr>
<td>Minimum distance between the outside facing edges of a notch drawn in an object</td>
<td>minNotch</td>
<td>Compactor, Layout Editor, Diva</td>
<td>spacingRules</td>
</tr>
<tr>
<td>Distance between objects drawn on the specified layer or layers</td>
<td>minSpacing</td>
<td>LAS, Compactor, Preview Silicon Ensemble, Preview Gate Ensemble, Diva</td>
<td>spacingRules</td>
</tr>
</tbody>
</table>
### Creating a Technology File: Generic Rules

**Minimum distance between the inside and outside edges of two overlapping objects**  

- **Rule**: minOverlap  
  - **Applications That Use It**: Diva  
  - **Section of Physical Rules Class**: spacingRules

**Distance by which an object must be enclosed by another object**  

- **Rule**: minEnclosure  
  - **Applications That Use It**: Compactor  
  - **Section of Physical Rules Class**: orderedSpacingRules
Sample Physical Rules Class

The following sample Physical Rules class illustrates the class and its subclasses (sections), along with the physical rules they define:

```plaintext
physicalRules(
  spacingRules(
    ; ( rule layer1 [layer2] value )
    ( minWidth cont 0.600000 )
    ( minSpacing cont 0.600000 )
    ...
    ( minSpacing metal1 via 0.600000 )
  ) ;end of spacingRules

  orderedSpacingRules(
    ; ( rule layer1 layer2 value )
    ( minEnclosure metal1 via 0.600000 )
    ( minEnclosure pimplt diff 0.300000 )
    ...
    ( minEnclosure prBnd cont 0.300000 )
  ) ;end of orderedSpacingRules

  mfgGridResolution(
    ;( g_resolution )
    ( 0.001000 )
  ) ;end of mfgGridResolution

) ; end of physicalRules
```

**Physical Rules class enclosure.** Follows Devices class. All subclasses must be specified within the parentheses of the class enclosure.

**Spacing Rules subclass.** Defines spacing rules in which the order of layers is not important. Specifies rule name, enclosing layer, layer within enclosing layer, and space that must separate the two layers. Data for each spacing rule must be enclosed in parentheses. All spacing rules must be enclosed within the parentheses of the subclass enclosure.

**Ordered Spacing Rules subclass.** Defines spacing rules in which the order of layers is important. Specifies rule name, enclosing layer, layer within enclosing layer, and space that must separate the two layers. All ordered spacing rules must be enclosed within the parentheses of the subclass enclosure.

**Grid-Snapping Resolution subclass.** Specifies that grid snapping must be a multiple of the value stated. The grid resolution value must be enclosed within the parentheses of the subclass enclosure.

**End of Physical Rules class enclosure.** All subclasses must be enclosed within the parentheses of the class enclosure.
Specifying Physical Rules

Technology file physical rules are required. As illustrated in the sample, the class must be specified by enclosing the subclass definitions within the `physicalRules()` class enclosure. The following paragraphs provide detailed information about specifying subclass data.

For more information about the `physicalRules` class, refer to the *Technology File and Display Resource File SKILL Reference Manual*. 
Specifying Spacing Rules: spacingRules()

The `spacingRules` subclass (section) of the Physical Rules class specifies the spacing rules, defining the amount of enclosure required when two layers overlap, in which the order of the layers is not important for your design.

**Spacing Rules subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Rule name.** Specify the name of the rule you are defining, selected from the list of available rules in the “Physical Rules” section of this chapter. The data for each rule must be enclosed within its own set of parentheses.

- **Layer name 1.** Specify the layer name or layer-purpose pair (enclosed in parentheses) for the enclosing layer.

- **Layer name 2.** Specify the layer name or layer-purpose pair (enclosed in parentheses) for the layer contained within the enclosing layer. If you do not specify this parameter, the rule applies to different objects on the same layer.

- **Value.** Specify the space, in user units, that must separate the two layers or different objects on the same layer.

```
spacingRules(
    ; ( rule layer1 [layer2] value )
    ( minWidth cont 0.600000)
    ( minSpacing cont 0.600000)
    ( minSpacing cont poly1 0.600000)
    ( minSpacing cont via 0.600000)
    ( minNotch diff 0.600000)
    ( minSpacing diff 0.600000)
    ( minWidth diff 0.600000)
    ( minNotch metall 0.600000)
    ( minWidth metall 0.600000)
    ( minSpacing metall 0.600000)
    ( minSpacing metall via 0.600000)
    ( defaultWidth metall 0.600000)
    ( maxWidth metall 0.600000)
) ; end of spacingRules
```

For more information about the `spacingRules` section, refer to the *Technology File and Display Resource File SKILL Reference Manual.*
Specifying Ordered Spacing Rules: orderedSpacingRules()

The orderedSpacingRules subclass (section) of the Physical Rules class specifies the spacing rules, defining the amount of enclosure required when two layers overlap, in which the order of the layers is important for your design.

Ordered Spacing Rules subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

- **Rule name.** Specify the name of the rule you are defining, selected from the list of available rules in the “Physical Rules” section of this chapter. The data for each rule must be enclosed within its own set of parentheses.

- **Layer name 1.** Specify the layer name or layer-purpose pair (enclosed in parentheses) for the enclosing layer.

- **Layer name 2.** Specify the layer name or layer-purpose pair (enclosed in parentheses) for the layer contained within the enclosing layer.

- **Value.** Specify the space, in user units, that must separate the two layers.

```plaintext
orderedSpacingRules(
  ; ( rule layer1 layer2 value )
  ( minEnclosure metal1 via 0.600000)
  ( minEnclosure pimplant diff 0.600000)
  ( minEnclosure prBoundary cont 0.600000)
  ( minEnclosure prBoundary diff 0.600000)
  ( minEnclosure prBoundary metal1 0.600000)
) ; end of orderedSpacingRules
```

For more information about the orderedSpacingRules section, refer to the *Technology File and Display Resource File SKILL Reference Manual.*
Specifying Grid-Snapping Rule: mfgGridResolution()

The mfgGridResolution subclass (section) of the Physical Rules class specifies that grid snapping must be a multiple of the value specified in the rule for your design.

Grid-Snapping Resolution subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

Value. Specify the grid-snapping value. The value must be enclosed in parentheses and must be within the parentheses of the subclass enclosure.

```plaintext
mfgGridResolution(
    ; ( value )
    ( 0.001000 )
) ; end of mfgGridResolution
```

For more information about the mfgGridResolution section, refer to the Technology File and Display Resource File SKILL Reference Manual.

Electrical Rules

Electrical rules specify electrical characteristics of the layers you use in your designs; for example, capacitance, density, resistance, and area capacitance.

The following table lists the rules you can set in your technology file and the applications that use them:

<table>
<thead>
<tr>
<th>Type of Electrical Rule</th>
<th>Rule Name</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>The loading capacitance</td>
<td>areaCap</td>
<td>Virtuoso XL, LAS, Preview Silicon Ensemble, Preview Gate Ensemble</td>
</tr>
<tr>
<td>of the layer in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>picofarads per square</td>
<td></td>
<td></td>
</tr>
<tr>
<td>micron</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The amount of current</td>
<td>currentDensity</td>
<td></td>
</tr>
<tr>
<td>that the default wire</td>
<td></td>
<td></td>
</tr>
<tr>
<td>width can carry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The capacitance between</td>
<td>edgeCapacitance</td>
<td></td>
</tr>
<tr>
<td>parallel objects</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

April 2001 90 Product Version 4.4.6
Electrical rules are specified in the Electrical Rules class, a generic rules class that can be used by all applications.

<table>
<thead>
<tr>
<th>Type of Electrical Rule</th>
<th>Rule Name</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>The fringe capacitance for the layer</td>
<td>parallelCap</td>
<td>Virtuoso XL, LAS,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Preview Silicon</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ensemble, Preview Gate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ensemble</td>
</tr>
<tr>
<td>The resistivity of the layer in ohms per</td>
<td>sheetRes</td>
<td></td>
</tr>
<tr>
<td>square</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Electrical rules are specified in the Electrical Rules class, a generic rules class that can be used by all applications.
Sample Electrical Rules Class

The following sample Electrical Rules class illustrates the class and its subclasses (sections), along with the electrical characterization rules they define:

```plaintext
electricalRules(  

    characterizationRules(  
      ; ( rule layer value )  
      ( areaCap  metal1  1.4e-4 )  
      ( edgeCap  metal1  2.000000 )  
      .  
      .  
      ( height  metal1  1.0000 )  
    ) ; end of characterizationRules  

    orderedCharacterizationRules(  
      ; ( rule layer1 layer2 value )  
      ( parallelCap  metal1  metal2  2.00 )  
      ( parallelCap  metal1  metal4  2.00 )  
      .  
      .  
      ( parallelCap  prBnd1  prBnd2  1.00 )  
    ) ; end of orderedCharacterizationRules  

  ) ; end of electricalRules  
```

**Specifying Electrical Rules**

Technology file electrical rules are required. As illustrated in the sample, the class must be specified by enclosing the subclass definitions within the `electricalRules()` class enclosure. The following paragraphs provide detailed information about specifying subclass data.

For more information about the `electricalRules` class, refer to the *Technology File and Display Resource File SKILL Reference Manual*. 
Specifying Electrical Characterization Rules: characterizationRules()

The characterizationRules subclass (section) of the Electrical Rules class specifies the electrical characterization rules in which the order of the layers is not important for your design.

Characterization Rules subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

- **Rule name.** Specify the name of the rule you are defining, selected from the list of available rules in the “Electrical Rules” section of this chapter. The data for each rule must be enclosed within its own set of parentheses.

- **Layer name.** Specify the layer name, layer number, or layer-purpose pair (enclosed in parentheses) for the layer on which to apply the rule. Recommendation: Organize sections according to layer when specifying rules for multiple layers.

- **Value.** Specify the value of the characterization rule, in user units.

```characterizationRules(
; ( rule layer value )
( areaCap metall1 1.4e-4 )
( currentDensity metall1 2.000000 )
( edgeCapacitance metall1 4.0e-11 )
( sheetRes metall1 0.040000 )
) ; end of characterizationRules
```

For more information about the characterizationRules section, refer to the Technology File and Display Resource File SKILL Reference Manual.
Specifying Electrical Characterization Rules for Ordered Layers: orderedCharacterizationRules()

The orderedCharacterizationRules subclass (section) of the Electrical Rules class specifies the electrical characterization rules in which the order of the layers is important for your design.

Ordered Characterization Rules subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

- **Rule name.** Specify the name of the rule you are defining, selected from the list of available rules in the “Electrical Rules” section of this chapter. The data for each rule must be enclosed within its own set of parentheses.

- **Layer name 1.** Specify the layer name, layer number, or layer-purpose pair (enclosed in parentheses) for the first layer on which to apply the rule. Recommendation: Organize sections according to layer when specifying rules for multiple layers.

- **Layer name 2.** Specify the layer name or layer-purpose pair (enclosed in parentheses) for the second layer on which to apply the rule.

- **Value.** Specify the value of the ordered characterization rule, in user units.

```plaintext
orderedCharacterizationRules(
    ; ( rule layer1 layer2 value )
    ( parallelCap metal1 metal2 2.00 )
    ( parallelCap metal3 metal4 1.00 )
) ; end of orderedCharacterizationRules
```

Creating a Technology File: Application-Specific Rules

Application-specific rules specified in the technology file control how physical design applications work. This chapter discusses the technology file classes that specify the following:

- “Virtuoso Layout Editor Rules” on page 96
- “Virtuoso XL Rules” on page 99
- “Virtuoso Compactor Rules” on page 103
- “Virtuoso Layout Synthesizer (LAS) Rules” on page 112
- “Place and Route Rules” on page 120

Other technology file classes define the following:

- Controls, described in Chapter 3, “Creating a Technology File: Control, Layer, and Device Definitions.”
- Layer definitions, described in Chapter 3, “Creating a Technology File: Control, Layer, and Device Definitions.”
- Devices, described in Chapter 3, “Creating a Technology File: Control, Layer, and Device Definitions.”
- Layer rules, described in Chapter 4, “Creating a Technology File: Generic Rules.”
- Physical rules, described in Chapter 4, “Creating a Technology File: Generic Rules.”
- Electrical rules, described in Chapter 4, “Creating a Technology File: Generic Rules.”
Virtuoso Layout Editor Rules

You use the Virtuoso® layout editor (VLE) to create a custom integrated circuit design or fine-tune a design that you have generated automatically with another tool. You can do the following with the layout editor:

- Draw and edit polygons, paths, rectangles, circles, ellipses, donuts, pins, and contacts in design layout cells
- Place cells into other cells to create hierarchical designs
- Create special “parameterized cells” (pcells) containing data that you want to modify quickly or that you want to set with Cadence® SKILL functions

The layout editor displays a tool called the Layer Selection Window (LSW), which is a palette of the layer-purpose pairs you use to draw your designs. You specify layout editor rules in the technology file to establish how layer-purpose pairs are displayed in the LSW. Layout editor rules define the following:

- Which layer-purpose pairs the LSW displays by default
- The order in which layer-purpose pairs are displayed

For more information about the Virtuoso Layout Editor, refer to the Virtuoso Layout Editor User Guide.
Sample Layout Editor Rules Class

The following sample Layout Editor Rules class illustrates the class and its subclass (section), along with the layout editor rules they define:

```plaintext
leRules(
  leLswLayers(
    ; ( layer         purpose )
    ( metal1         drawing )
    ( metal2         drawing )
    ( metal3         drawing )
    ( poly1          drawing )
    ( pwell          drawing )
    ( nimplant       drawing )
    ( diff           drawing )
  ) ; end of leLswLayers
)
) ; end of leRules
```

**Layout Editor Rules class enclosure.** Place anywhere after the Layer Definitions class; usually follows the Devices class. The one subclass must be specified within the parentheses of the class enclosure.

**LSW Display subclass.** Lists layer-purpose pairs in the order in which they are to be displayed in the LSW. If no `leLswLayers` subclass is defined, the LSW displays layers in priority order as specified in the `techLayerPurposePriorities` subclass of the Layer Definitions class. Each layer-purpose pair must be enclosed in parentheses. The entire list of layer-purpose pairs must be contained within the parentheses of the subclass enclosure.

**End of Layout Editor Rules class enclosure.** The subclass must be inside the parentheses of the class enclosure.

Specifying Layout Editor Rules

Layout editor rules are optional when using the layout editor and are not applicable to any other application. As illustrated in the sample, the class must be specified by enclosing the subclass definitions within the `leRules()` class enclosure. The following paragraphs provide detailed information about specifying subclass data.

For more information about the `leRules` class, refer to the Technology File and Display Resource File SKILL Reference Manual.

Specifying LSW Layers Rules: leLswLayers()

The `leLswLayers` subclass (section) of the Layout Editor Rules class lists layer-purpose pairs in the order in which they are to be displayed in the LSW. If you do not define `leLswLayers`, the layout editor displays layers in priority order as specified in the
Creating a Technology File: Application-Specific Rules

**LSW Layers subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

Layer-Purpose Pair (Layer name. Purpose name.) Specify the layer-purpose pairs in the order in which you want them displayed in the LSW. You must list layer-purpose pairs; you cannot list layer names alone. Each layer-purpose pair must be unique, must be enclosed in parentheses, and must be defined in the Layer Definitions class.

```plaintext
leLswLayers(
; ( layerName purpose )
( nwell drawing )
( nwell net )
( nwell pin )
( pwell drawing )
( pwell net )
( pwell pin )
) ; end of leLswLayers
```

The following table summarizes which layer-purpose pairs the LSW displays itself and in its Set Valid Layer form when you do or do not specify `leLswLayers`:

<table>
<thead>
<tr>
<th><code>leLswLayers Specified in Technology File</code></th>
<th>Layer-Purpose Pairs Listed in LSW</th>
<th>Layer-Purpose Pairs Listed in LSW's Set Valid Layer Form</th>
<th>Order of Layer-Purpose Pairs Listed in LSW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>All <em>valid</em> layers specified in <code>leLswLayers</code></td>
<td>All layers specified in <code>leLswLayers</code></td>
<td>Same as the order specified in <code>leLswLayers</code></td>
</tr>
<tr>
<td>No</td>
<td>All <em>valid</em> layers listed in the technology library</td>
<td>All layers listed in the technology library</td>
<td>Priority order as specified in the Layer Definitions class, <code>techLayerPurposePriorities subclass</code></td>
</tr>
</tbody>
</table>

Notice that in either case, only valid layers are listed in the LSW. If no layers are valid, no layers are displayed in the LSW.
Note: While the technology file entries determine the default LSW layer display, the LSW itself provides commands that let you modify whether layers are selectable and visible as you work.


Virtuoso XL Rules

You use the Virtuoso layout accelerator (Virtuoso XL) with the layout editor to generate custom layouts from schematics or to edit layouts that have defined connectivity. Virtuoso XL continuously monitors connections between components in the layout and compares them with connections in the schematic. Virtuoso XL uses an online “extractor” to monitor the layers in a design and identify shorts in connectivity by looking for layers that cannot overlap. You use Virtuoso XL to view incomplete nets, shorts, invalid connections, and illegal overlaps in your design.

You specify Virtuoso XL rules to provide the information Virtuoso XL needs to monitor the connectivity in a layout. Virtuoso XL rules specify the following types of information:

■ Routing layers you want Virtuoso XL to monitor
■ Layers that cannot overlap in a design

You must set the Virtuoso XL rules in the technology file before you can use the tool to connect the elements of a design.

For more information about Virtuoso XL, refer to the Virtuoso Layout Accelerator User Guide.
Sample Virtuoso XL Rules Class

The following sample Virtuoso XL Rules class illustrates the class and its subclasses (sections), along with the Virtuoso XL rules they define.

```plaintext
lxRules()

lxExtractLayers()
; layer...
pwell ndiff pdiff poly1 cont metall1
via metal12 (metal1 drawing1)
(metal12 drawing1) via2 metal13
); end of lxExtractLayers

lxNoOverlapLayers()
; ( layer1 layer2 )
( poly1 ndiff )
( poly1 pdiff )
( via via2 )
( metal2 drawing1 )
); end of lxNoOverlapLayers

lxMPPTemplates()
; ( t_mppTemplateName l_template )
...;
); end of lxMPPTemplates

); end of lxRules

Virtuoso XL Rules class enclosure. Place anywhere after the Layer Definitions class; usually follows the Layout Editor Rules class. All subclasses must be specified within the parentheses of the class enclosure.

Extract Layers subclass. Lists layers for the online extractor to monitor. All extract layer data must be contained within the parentheses of the subclass enclosure.

No Overlap Layers subclass. Lists layers that cannot overlap in a Virtuoso XL design. Must follow lxExtractLayers. Layers listed here must also be listed in lxExtractLayers. All no overlap layer data must be contained within the parentheses of the subclass enclosure.

MPP Templates subclass. Defines ROD multipart path templates. All template definitions must be contained within the parentheses of the subclass enclosure.

End of Virtuoso XL Rules class enclosure. All subclasses must be inside the parentheses of the class enclosure.

Specifying Virtuoso XL Rules

Virtuoso XL Rules are required when you are using Virtuoso XL. As illustrated in the sample, the class must be specified by enclosing the subclass definitions within the lxRules() class enclosure. The following paragraphs provide detailed information about specifying subclass data.

For more information about the lxRules class, refer to the Technology File and Display Resource File SKILL Reference Manual.
Specifying Extract Layers Rules: lxExtractLayers()

The lxExtractLayers subclass (section) of the Virtuoso XL Rules class specifies which layers in your design are to be monitored by the online extractor.

Extract Layers subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

```
xExtractLayers(
    ; ( layer... )
    ( pwell ndiff pdiff poly1 cont metal1 via metal2
      (metal1 drawing1) (metal2 drawing1) via2 metal3 )
); end lxExtractLayers
```

Layer names. Specify the layer name or layer-purpose pair (enclosed in parentheses) for each layer to be monitored by the online extractor. Listing just the layer name forces the extractor to monitor all layer-purpose pairs with that layer name. Layer-purpose pairs must already be defined in the Layer Definitions class. The entire list of layer names must be enclosed in parentheses.

Specifying No Overlap Layers Rules: lxNoOverlapLayers()

The lxNoOverlapLayers subclass (section) of the Virtuoso XL Rules class specifies the layers in your design that cannot overlap.

No Overlap Layers subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

Layer name 1. Specify the layer name or layer-purpose pair (enclosed in parentheses) for the first of two layers that cannot overlap. The layer must also be listed in lxExtractLayers. Each pair of layers must be separated by a space and enclosed in parentheses.

Layer name 2. Specify the layer name or layer-purpose pair (enclosed in parentheses) for the second of two layers that cannot overlap. The layer must also be listed in lxExtractLayers. Each pair of layers must be separated by a space and enclosed in parentheses.

```
lxNoOverlapLayers(  
; ( layer1      layer2   )  
( poly1       ndiff    )  
( poly1       pdiff    )  
( via         via2     )  
( metal2      drawing1 )  
) ; end of lxNoOverlapLayers
```

Specifying Multipart Path Templates: lxMPPTemplates()

The `lxMPPTemplates` subclass (section) of the Virtuoso XL Rules class defines a template or series of templates for relative object design (ROD) multipart paths (MPPs). A multipart path is a single ROD object consisting of one or more parts at level zero in the hierarchy on the same or on different layers. The purpose of an MPP template is to let you create MPPs in layout cellviews using predefined values. You can define any number of MPP templates in the `lxMPPTemplates` subclass.

**MPP Templates subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

```plaintext
lxMPPTemplates(
    ; ( t_mppTemplateName l_template)
) ;end lxMPPTemplates
```

For more information about the `lxMPPTemplates` subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual*.

**Virtuoso Compactor Rules**

You use the Virtuoso compactor to optimize symbolic designs by packing symbolic objects as closely as possible. The compactor also helps you create layouts that adhere to design and electrical rules.

You specify compactor rules to provide the information the compactor needs to optimize a layout. Compactor rules specify the following types of information:

- Layer usage with the compactor
- Symbolic wires and associated constraints that you use in your designs
- Design rules that control compactor functions

For more information about the compactor, refer to the *Virtuoso Compactor Reference Manual*. 
Sample Compactor Rules Class

The following sample Compactor Rules class illustrates the class and its subclasses (sections), along with the compactor rules they define:

```lisp
compactorRules(

  compactorLayers(
    ; ( layer usage )
    ( diff "diffusion" )
    ( poly1 "conduction" )
    ...
    ( bndry "cellBoundary" )
  ) ; end of compactorLayers

symWires(
  ; ( name ( layer purpose)
    ( PDIFF ( "diff" "drawing")
    ; [[( impLayer impSpacing )]
        ( "implant" "drawing") 0.2
    ; [[ defaultW minW maxW ]]
      ( 0.6 nil nil )
    ; [[ regionName regionLayer ]]
      ( "inside" ( "nwell" "drawing" ) ) 100.0 
    ...
  ) ; end of symWires

symRules(
  ; ( drc
    ; ( layer1 [ purpose1 [cell1] [view1]])
      ( drc
        ( "pimplant" "drawing" "PTAP" schematic )
    ; [[ layer2 [ purpose2 [cell2] [view2]]])
      ( "diff" "drawing" "NTR" schematic)
    ; ( ruleType < value ) [ modifier1 ] [ modifier2 ]
      ( sep < 0.900000) "sameNet" "vertical"
    ...
  ) ;end symRules

) ;end of compactorRules
```

Compactor Rules class enclosure. Place anywhere after the Devices class (it is typically placed after the Layout Editor Rules class) and before the LAS Rules class. All subclasses must be specified within the parentheses of the class enclosure.

Compactor Layers subclass. Lists layers and specifies how the compactor is to use them. All layer data must be contained within the parentheses of the subclass enclosure.

Wires subclass. Defines the wires used in your design. Used by the compactor and the layout synthesizer. All wire data must be contained within the parentheses of the subclass enclosure.

Compactor Design Rules subclass. Lists design rules the compactor follows when it compacts your design. This subclass is usually located at the end of the Compactor Rules class. All compactor design rule data must be contained within the parentheses of the subclass enclosure.

End of Compactor Rules class enclosure. All subclasses must be inside the parentheses of the class enclosure.
Specifying Compactor Rules

Compactor Rules are required when you are using the Virtuoso compactor. As illustrated in the sample, the class must be specified by enclosing the subclass definitions within the `compactorRules()` class enclosure. The following paragraphs provide detailed information about specifying subclass data.

For more information about the `compactorRules` class, refer to the *Technology File and Display Resource File SKILL Reference Manual*.

Specifying Compactor Layer Usage: `compactorLayers()`

The `compactorLayers` subclass (section) of the Compactor Rules class specifies how the compactor treats layers in your design.

Compactor Layers subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

```
compactorLayers(
    ; ( layer usage )
    ( diff  "diffusion" )
    ( poly1 "conduction" )
    ( metal "conduction" )
    ( metal2 "conduction" )
    ( metal3 "conduction" )
    ( ( via  pin ) "via" )
    ( ( via2 pin ) "via" )
    ( pwell "well" )
    ( pimplant "implant" )
    ( hardFence "hardfence" )
    ( softFence "softfence" )
    ( prBoundary "cellBoundary" )
)
```

For more information about the `compactorLayers` subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual*. For more information about the layer usage selected by the keywords, refer to the *Virtuoso Compactor Reference Manual*.
Defining Symbolic Wires: symWires()

The compactor can stretch and insert jogs in the wires or paths in a layout design. You must define the wires in the technology file so the Compactor knows which wire to use. LAS also uses these wires to generate layouts.

When defining symWires, you must specify a layer name and purpose, for which you can apply the following optional constraints to the wire:

- Implant layer and enclosure spacing
- Width settings: minimum, maximum, or default
- Legal region
- Wire length minimization weight

The symWires subclass (section) of the Compactor Rules class defines wires for the compactor.
Wires subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

Wire name. Specify the name of the wire to update or create.

Layer-Purpose Pair (Layer name. Purpose name.). Specify the name of the layer for the wire, followed by its purpose. The layer-purpose pair must be enclosed in parentheses.

Implant layer constraints. You must specify these constraints or, to define wire width data without defining implant data, specify nil. To specify implant layer constraints, specify the implant layer and spacing as defined in the following:

Implant layer name. Specify the name of the implant layer or layer-purpose pair (enclosed in parentheses) to which to apply the constraint. Default purpose: drawing.

Implant spacing. Specify the distance, in user units, by which the implant layer must enclose the wire layer.

Note: Continued in following figure.
Wire width specifications. You must specify, in parentheses, a value for at least one of the following widths, in user units (for any parameter for which you specify no value, you must specify nil):

Default width. Specify the default wire width, in user units.

Minimum width. Specify the minimum allowable wire width, in user units.
Default: 0.

Maximum width. Specify the maximum allowable wire width, in user units.
Default: infinity.

Note: Continued in following figure.
Legal region. To define the legal region, specify the region name and region layer as defined in the following:

Legal region name. Specify the legal region, either inside or outside.

Legal region layer. Specify the layer name or layer-purpose pair (enclosed in parentheses) for the layer that is the object of the legal region layer. Default: drawing.

Weighting. Specify the weighting, as any positive number, indicating the relative priority to the compactor for minimizing the length of the wire.

symWires(
 ; (name ( layer purpose) [ (impLayer impSpacing)])
 ( PDIFF ( "diff" "drawing") ("implant" "drawing") 0.2 )
 ; [( defaultW minW maxW )]
 ( 0.6 nil nil )
 ; [( regionName regionLayer )]
 ( "inside" ( "nwell" "drawing" ) ) 100.0 )
);


Setting Design Rules for Compaction: symRules

Design rules specific to compactor functions differ from Physical Rules in that you can restrict rules to objects on the same net or on different nets, or to layers on specific devices or pcells.

You can specify the following constraints to control design compaction:

■ Minimum distance between outside facing edges of two objects on the same layer
■ Minimum distance between outside facing edges of two objects on different layers
■ Minimum enclosure distance between objects on different layers

The symRules subclass (section) of the Compactor Rules class defines design rules for the compactor.
Compactor Design Rules subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

**drc.** Always specify the keyword *drc* to indicate that design rules follow.

**Layer name.** Specify the layer name for the layer for which you are defining the compactor rule. Each layer specified must be defined in the Layer Definitions class and must be assigned a compactor keyword in the *compactorLayers* subclass.

**Purpose name.** Specify the purpose for the layer for which you are defining the Compactor rule. If you specify a cell and view, you must specify a layer purpose.

**Cell name.** Specify the name of the master cell of the component.

**View name.** Specify the view name of the component. If you specify a cell name but no view name, the default is *symbolic*.

**Second layer data.** The parameters for the second layer are identical to those for the first layer. If not specified, the rule applies to objects drawn on the same layer and instances of the same device.

### Example

```plaintext
symRules(
    ( drc
        ( layer1 [ purpose1 [cell1] [view1]]
            ( drc
                ( "pimplant" "drawing" "PTAP" schematic)
                ;
                ( layer2 [ purpose2 [cell2] [view2]]
                    ( "diff" "drawing" "NTR" schematic)
                    ;
                    ( ruleType < value) [ modifier1 ] [ modifier2 ]
                        ( sep < 0.900000) "sameNet" "vertical"

    ) ;end symRules
```

**Note:** Continued in following figure.
Rule type. Specify the type of rule, either separation (sep) or enclosure (enc).

Value. Specify the minimum separation or enclosure distance allowed or the keyword dontcare.

Modifier1. Specify a modifier (sameNet or diffNet) to place a restriction on the rule.

Modifier2. Specify a modifier (horizontal or vertical) to place a restriction on the rule.

```
symRules(
    ( drc
        ( drc
            ( "pimplant" "drawing" "PTAP" schematic)
            ( layer1 [ purpose1 [cell1] [view1]])
        ( drc
            ( "diff" "drawing" "NTR" schematic)
            ( layer2 [ purpose2 [cell2] [view2]])
        ( ruleType < value) [ modifier1 ] [ modifier2 ]
            ( sep < 0.900000) "sameNet" "vertical"
    )
) ; end of symRules
```

Virtuoso Layout Synthesizer (LAS) Rules

You use the Virtuoso layout synthesizer (LAS) to generate uncompacted physical layouts from schematics, transistor netlists, or autoLayout views. You can do the following with LAS:

- Generate layouts from hierarchical circuits that can be flattened to the level of individual transistors
- Build customized cells and digital random logic blocks
- Create standard cells for standard cell libraries and datapath libraries

You specify layout synthesizer rules to define the layers, wires, devices, and special properties LAS uses to generate layout designs. You must set the layout synthesizer rules in the technology file before you can use LAS to generate layout designs.

For more information about the Virtuoso layout synthesizer, refer to the Virtuoso Layout Synthesizer User Guide.
Sample Layout Synthesizer Rules Class

The following sample Layout Synthesizer Rules class illustrates the class and its subclasses (sections), along with the rules they define:

```plaintext
lasRules(

  lasLayers(
    ; ( layer usage )
    ( ndiff "ndiffLayer" )
    ( pdiff "pdiffLayer" )
    ( pwell "pwellLayer" )
    ( poly1 "polyLayer" )
    ( metal1 "metal1Layer" )
    ( metal2 "metal2Layer" )
    .
    .
    ( metal3 "metal3Layer" )
  ); end of lasLayers

  lasDevices(
    ; ( cellview LASDevName )
    ( (NTR symbolic) "ntr" )
    ( (PTR symbolic) "ptr" )
    ( (M1_P symbolic) "pd_m1" )
    ( (M1_N symbolic) "nd_m1" )
    ( (M1_POLY1 symbolic) "poly_m1" )
    ( (M1_M2 symbolic) "m1_m2" )
    ( (PTAP symbolic) "psub_m1" )
    ( (NTAP symbolic) "nsub_m1" )
    ( (poly1_T symbolic) "poly_pin" )
    ( (metal1_T symbolic) "m1_pin" )
    ( (metal2_T symbolic) "m2_pin" )
  ); end of lasDevices

  lasWires(
    ; ( wireName wireType )
    ( "ndiff" "ndiffWire" )
    ( "pdiff" "pdiffWire" )
    .
    .
  ); end of lasWires
```

**Layout Synthesizer Rules class enclosure.** Must be placed anywhere after the Compactor Rules class. All subclasses must be specified within the parentheses of the class enclosure.

**LAS Layers subclass.** Lists layers and how LAS uses them. Data for each layer must be enclosed in parentheses. All layer specifications must be enclosed within the parentheses of the subclass enclosure.

**LAS Devices subclass.** Lists the devices LAS uses when it automatically synthesizes a design. Specifies cell, view, and LAS device name. Data for each cellview must be enclosed in parentheses, and the cellview and device name must be within parentheses. All LAS device specifications must be enclosed within the parentheses of the subclass enclosure.

**LAS Wires subclass.** Lists wires and how LAS uses them. Data for each wire must be enclosed in parentheses. All wire specifications must be enclosed within the parentheses of the subclass enclosure.
Specifying Layout Synthesizer Rules

Layout synthesizer rules are required when you are using the Virtuoso layout synthesizer. As illustrated in the sample, the class must be specified by enclosing the subclass definitions within the lasRules() class enclosure. The following paragraphs provide detailed information about specifying subclass data.

For more information about the lasRules class, refer to the Technology File and Display Resource File SKILL Reference Manual.

Specifying Layer Usage: lasLayers()

LAS creates and routes devices when it generates a layout. To accomplish this, LAS needs to know which layers you want it to use and how you want it to use them.

The lasLayers subclass (section) of the Layout Synthesizer Rules class specifies how LAS treats layers in your design.

```
lasProperties(  
; ( propName     propValue )  
( "lasSwitchViewList"   "schematic symbolic" )  
( "lasPrimitiveList"    "pmos nmos PTR NTR
                          HPTR HNTF" )  
( "lasPseudoElementList" "vdd gnd ndiff_T
                        pdiff_T poly2_T" )  
( "lasNSpiceCellView"    "nmos cdl" )  
( "lasPSpiceCellView"    "pmos cdl" )  
( "lasMfactorSplit"      nil )  
( "lasNetlistNtr"        "hnmos nmos" )  
( "lasNetlistPtr"        "hpmos pmos" )  
( "lasLayoutNtr"         "HNTR NTR" )  
( "lasLayoutPtr"         "HPTR PTR" )  
) ; end of lasProperties

) ; end of lasRules
```
The following table lists all of the LAS layer keywords. In some cases, you can use an alternative generic keyword instead of the more specific primary keyword.

<table>
<thead>
<tr>
<th>LAS Usage</th>
<th>Layer Keyword</th>
<th>Alternative Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffusion</td>
<td>ndiffLayer, pdiffLayer</td>
<td>diffLayer</td>
</tr>
<tr>
<td>Well</td>
<td>nwellLayer, pwellLayer</td>
<td></td>
</tr>
<tr>
<td>Poly</td>
<td>polyLayer</td>
<td></td>
</tr>
<tr>
<td>Metal</td>
<td>met1Layer, met2Layer, met3Layer</td>
<td></td>
</tr>
<tr>
<td>Metal1 power and ground</td>
<td>met1PowerLayer, met1GroundLayer</td>
<td>met1PGLayer</td>
</tr>
<tr>
<td>Metal2 power and ground</td>
<td>met2PowerLayer, met2GroundLayer</td>
<td>met2PGLayer</td>
</tr>
<tr>
<td>Ring</td>
<td>nringLayer, pringLayer</td>
<td>ringLayer</td>
</tr>
</tbody>
</table>

Specifying Devices That Can Be Placed: lasDevices()

LAS often needs to place instances of devices in a generated design. To accomplish this, LAS needs to know which device cellviews to place.

The lasDevices subclass (section) of the Layout Synthesizer Rules class specifies device cellviews for LAS to use in your design.

**LAS Devices subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

```
lasDevices(
    ; ( cellview LAS name )
    ( ( NTR symbolic ) "ntr" )
    ( ( PTR symbolic ) "ptr" )
    ( ( M1_P symbolic ) "pd_m1" )
    ( ( M1_N symbolic ) "nd_m1" )
    ( ( M1_POLY1 symbolic ) "poly_m1" )
    ( ( M1_M2 symbolic ) "m1_m2" )
    ( ( PTAP symbolic ) "psub_m1" )
    ( ( poly1_T symbolic ) "poly_pin" )
    ( ( metal1_T symbolic ) "m1_pin" )
    ( ( metal2_T symbolic ) "m1_pin" )
) ; end of lasDevices
```

The following table lists all of the LAS device keywords. In some cases, you can use an alternative generic keyword instead of the more specific primary keyword.

<table>
<thead>
<tr>
<th>LAS Usage</th>
<th>Device Keyword</th>
<th>Alternative Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOS transistors</td>
<td>ntr, ptr</td>
<td></td>
</tr>
<tr>
<td>Required pins</td>
<td>poly_pin, m1_pin, pd_m1, nd_m1, poly_m1</td>
<td></td>
</tr>
<tr>
<td>Optional pins</td>
<td>m2_pin, m3_pin</td>
<td></td>
</tr>
<tr>
<td>Special-purpose pins</td>
<td>nring_pin, pring_pin</td>
<td>ring_pin</td>
</tr>
</tbody>
</table>
Specifying Wires for LAS Use: lasWires()  

LAS places implant and ring wires when it generates a layout design. To place wires, LAS needs to know which wires you want to use for what purpose.

The lasWires subclass (section) of the Layout Synthesizer Rules class defines the wires for LAS to use in your design.

**LAS Wires subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Wire name.** Specify the wire name. Each wire specified must be defined in the symWires subclass of the Compactor Rules class.

- **Wire type.** Specify the keyword indicating the LAS wire type. See the table following this figure for the LAS wire keywords.

```plaintext
lasWires(
  ; ( wireName wireType )
  ( "ndiff" "ndiffWire" )
  ( "pdiff" "pdiffWire" )
  .
  .
) ; end of lasWires
```
The following table lists the LAS wire keywords. In some cases, you can use an alternative generic keyword instead of the more specific primary keyword.

<table>
<thead>
<tr>
<th>LAS Usage</th>
<th>Wire Keyword</th>
<th>Alternative Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffusion implant wires</td>
<td>ndiffWire, pdiffWire</td>
<td>diffWire</td>
</tr>
<tr>
<td>Guard ring wires</td>
<td>nringWire, pringWire</td>
<td>ringWire</td>
</tr>
</tbody>
</table>

For more information about the lasWires subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual*.

**Specifying LAS Properties: lasProperties()**

LAS provides properties that you can set to control how it generates a layout.

The lasProperties subclass (section) of the Layout Synthesizer Rules class specifies the properties for LAS to use in your design.

**LAS Properties subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

```
lasProperties(
    ; ( propName propValue )
    ( "lasSwitchViewList" "schematic symbolic" )
    ( "lasPrimitiveList" "pmos nmos PTR NTR HPTR HNTR" )
    ( "lasPseudoElementList" "vdd gnd ndiff_T pdiff_T poly2_T" )
    ( "lasNSpiceCellView" "nmos cd1" )
    ( "lasPSpiceCellView" "pmos cd1" )
    ( "lasMfactorSplit" nil )
    ( "lasNetlistNtr" "hnmos nmos" )
    ( "lasNetlistPtr" "hpmos pmos" )
    ( "lasLayoutNtr" "HNTR NTR" )
    ( "lasLayoutPtr" "HPTR PTR" )
) ; end of lasProperties
```
The following table lists the LAS-provided properties and their types and default values.

<table>
<thead>
<tr>
<th>Valid LAS Property Name</th>
<th>Data-type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lasBulkNameList</td>
<td>String</td>
<td>&quot;B b BULK Bulk bulk&quot;</td>
</tr>
<tr>
<td>lasDrainNameList</td>
<td>String</td>
<td>&quot;D d DRAIN Drain drain&quot;</td>
</tr>
<tr>
<td>lasGateNameList</td>
<td>String</td>
<td>&quot;G g GATE Gate gate&quot;</td>
</tr>
<tr>
<td>lasInstTypePropName</td>
<td>String</td>
<td>&quot;subtype&quot;</td>
</tr>
<tr>
<td>lasLayoutNtr</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>lasLayoutPtr</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>lasLengthName</td>
<td>String</td>
<td>&quot;l&quot;</td>
</tr>
<tr>
<td>lasMasterTypePropName</td>
<td>String</td>
<td>&quot;instNamePrefix&quot;</td>
</tr>
<tr>
<td>lasMfactorName</td>
<td>String</td>
<td>&quot;m&quot;</td>
</tr>
<tr>
<td>lasMfactorSplit</td>
<td>Boolean</td>
<td>t</td>
</tr>
<tr>
<td>lasNSpiceCellView</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>lasNetlistNtr</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>lasNetlistPtr</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>lasPSpiceCellView</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>lasPrimitiveList</td>
<td>String</td>
<td>&quot;mos pmos nmos pfet nfet&quot;</td>
</tr>
<tr>
<td>lasPseudoElementList</td>
<td>String</td>
<td>&quot;vdd gnd Asize Bsize Csize ASIZE BSIZE CSIZE&quot;</td>
</tr>
<tr>
<td>lasReadCDFParam</td>
<td>Boolean</td>
<td>t</td>
</tr>
<tr>
<td>lasSourceNameList</td>
<td>String</td>
<td>&quot;S s SOURCE Source source&quot;</td>
</tr>
<tr>
<td>lasSwitchViewList</td>
<td>String</td>
<td>&quot;schematic gate.sch cmos.sch netlist symbol symbolic&quot;</td>
</tr>
<tr>
<td>lasVddName</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>lasVddNameList</td>
<td>String</td>
<td>&quot;</td>
</tr>
<tr>
<td>lasVssName</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>lasVssNameList</td>
<td>String</td>
<td>&quot;</td>
</tr>
</tbody>
</table>
Place and Route Rules

You use the place-and-route tools to perform design place and route functions as follows:

- *Preview Silicon Ensemble™* software performs layout for very large standard cell-based integrated circuits. It supports two- through six-layer routing.
- *Preview Gate Ensemble®* software performs place and route functions for gate array design.

You specify Place and Route Rules to provide the information the place-and-route tools need to optimize a layout. Place and Route Rules specify the following types of information:

- Routing layers and their order of placement
- Devices used as vias and the rules that apply to them
- Stackable contact vias
- Master slice layers
- Rules for placing vias
- Rules for generating vias
- Rules for closing gaps on 90-degree turns on via layers
- Rules for routing with nondefault wire widths
- Minimum allowable spacing between geometries on different nets
- Distance between placement and routing grids
- Overlap boundaries, where cells cannot overlap

For more information about the Preview place-and-route tools, refer to the following manuals:

- *Preview Silicon Ensemble Reference*
- *Preview Gate Ensemble Reference*
Sample Place and Route Rules Class

The following sample Place and Route Rules class illustrates the class and its subclasses (sections), along with the rules they define.

```plaintext
prRules(
    prRoutingLayers(
        ; (layer direction)
        (poly1 "vertical")
        (metal1 "horizontal")
        (metal2 "vertical")
        (metal3 "horizontal")
    ) ; end of prRoutingLayers

    prViaTypes(
        ; (device cellViewName viaType)
        (M2_M1 symbolic) "default"
        (M3_M2 symbolic) "default"
        (ND1M2_M1 symbolic) "NDrule1"
        (ND1M3_M2 symbolic) "NDrule1"
        (ND2M2_M1 symbolic) "NDrule2"
        (ND2M3_M2 symbolic) "NDrule2"
    ) ; end prViaTypes

    prStackVias(
        ; (layer... via via2)
        (poly1 drawing) poly2
    ) ; end of prStackVias

    prMastersliceLayers(
        ; (layer... diff poly1)
    ) ; end prMastersliceLayers
```

**Place and Route Rules class enclosure.** Must be placed anywhere after the Devices class. All subclasses must be specified within the parentheses of the class enclosure.

**Routing Layers subclass.** Lists routing layers in the order of placement, with layers closest to the substrate listed first. Data for each layer must be enclosed in parentheses. All layer specifications must be enclosed within the parentheses of the subclass enclosure.

**Via Types subclass.** Lists the devices that the place and route tools use as vias; also specifies the rules that apply to the devices. Data for each device must be enclosed in parentheses, and the device name and cellview must be within parentheses. All via type specifications must be enclosed within the parentheses of the subclass enclosure.

**Stack Vias subclass.** Lists pairs of contact via layers that can be stacked in Preview Gate Ensemble software and Silicon Ensemble software. The list of layers must be enclosed in parentheses. All stack layer data must be enclosed within the parentheses of the subclass enclosure.

**Master Slice Layers subclass.** Lists master slice layers, from lowest to highest, for use by Preview Gate Ensemble software and Silicon Ensemble software. The list of layers must be enclosed in parentheses. All master slice layer data must be enclosed within the parentheses of the subclass enclosure.
Via Placement Rules subclass. Defines rules for placing vias with SROUTE or the Preview PowerRoute options of Preview Gate Ensemble software and Preview Silicon Ensemble software. Each rule definition must be enclosed in parentheses and lists the rule name, the list of devices to use, and the routing situation of the two routing layers. All rule data must be within the parentheses of the subclass enclosure.

```
prViaRules(
    ; ( ruleName     viaName
        ; layer1  dir1     ( wMin wMax overhang metalOverHang )
        ; layer2  dir2     ( wMin wMax overhang metalOverHang )
        ( "viaSP21"      (M2_M1)
            metal1  "vertical" ( .6 1.8 _NA_ _NA_ )
            metal2  "horizontal" ( .6 1.8 _NA_ _NA_ )
        )
    ) ; end of prViaRules
```

Via Generation Rules subclass. Defines rules for generating vias with SROUTE or the Preview PowerRoute options of Preview Gate Ensemble software and Preview Silicon Ensemble software. Each rule definition must be enclosed in parentheses and lists the rule name, the dimensions of the via to create, and the routing situation of the two routing layers. All rule data must be within the parentheses of the subclass enclosure.

```
prGenViaRules(
    ; ( ruleName     layer
        ; ( width  length  xPitch  yPitch  resistance )
        ; layer1  dir1     ( wMin wMax overhang metalOverHang )
        ; layer2  dir2     ( wMin wMax overhang metalOverHang )
        ( viagen21    via
            ( .6  .6 1.2 1.2 _NA_ _NA_ )
            metal1  "horizontal" ( .6 2.0 .6_ .6 )
            metal2  "vertical"   ( .6 2.0 .6  .6 )
        )
    ) ; end of prGenViaRules
```

Turn Via Rules subclass. Defines rules for closing a gap on a 90-degree turn on a via layer when placing with SROUTE or the Preview PowerRoute options of Preview Gate Ensemble software and Preview Silicon Ensemble software. Each rule definition must be enclosed in parentheses. All rule data must be enclosed within the parentheses of the subclass enclosure.

```
prTurnViaRules(
    ; ( ruleName     layer ( directionList )
        ( "TURN1"       metal1 ("vertical" "horizontal" )
        ( "TURN2"       metal2 ("vertical" "horizontal" )
        ( "TURN3"       metal3 ("vertical" "horizontal" )
    ) ; end of prTurnViaRules
```
Nondefault Rules subclass.

Defines rules for routing with nondefault wire widths with SROUTE or the Preview PowerRoute options of Preview Gate Ensemble software and Preview Silicon Ensemble software. Each rule definition must be enclosed in parentheses. All rule data must be enclosed within the parentheses of the subclass enclosure.

Routing Pitch subclass. Used by Silicon Ensemble software only.

Defines the routing pitch (the minimum allowable spacing, center to center) between two regular geometries on different nets. Data for each layer must be enclosed in parentheses. All routing pitch specifications must be enclosed within the parentheses of the subclass enclosure.

Routing Offset subclass. Used by Silicon Ensemble software only.

Defines the routing offset, or the distance between the placement grid and the routing grid when there is a routing grid between two placement grids. Data for each layer must be enclosed in parentheses. All routing offset specifications must be enclosed within the parentheses of the subclass enclosure.

Overlap Layer subclass.

Defines the overlap layer or layers used to display the overlap boundary for cells. The overlap boundary indicates where cells cannot overlap. The overlap layers specified must be enclosed within the parentheses of the subclass enclosure.

End of Place and Route Rules class enclosure. All subclasses must be inside the parentheses of the class enclosure.
Specifying Place and Route Rules

Place and Route Rules are required when you are using the Preview place-and-route tools (Preview Silicon Ensemble and Preview Gate Ensemble). As illustrated in the sample, the class must be specified by enclosing the subclass definitions within the prRules() class enclosure. The following paragraphs provide detailed information about specifying subclass data.

For more information about the prRules class, refer to the Technology File and Display Resource File SKILL Reference Manual.

Specifying Routing Layers: prRoutingLayers()

The prRoutingLayers subclass (section) of the Place and Route Rules class defines the routing direction of layers the place and route tools use for routing.

Routing Layers subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

```
prRoutingLayers(
    ; ( layer direction )
    ( poly1 "vertical" )
    ( metal1 "horizontal" )
    ( metal2 "vertical" )
    ;
) ; end of prRoutingLayers
```

Specifying Vias: `prViaTypes()`

The `prViaTypes` subclass (section) of the Place and Route Rules class defines the vias for the place and route tools use during automatic routing.

**Via Types subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Device name and cellview name.** Specify the device name and cellview name, in parentheses and separated by a space, as follows:
  - **Device name.** Specify the device name. Each device specified must be defined in the Devices class.
  - **Cellview name.** Specify the cellview name.

- **Via type.** Specify the name of the rule indicating how the place and route tools are to use the device. Specify either `default` (selects default width routing for all tools) or a rule name defined in the `prNonDefaultRules` subclass.

```prViaTypes()

; ( ( device cellViewName ) viaType  )
(   ( M2_M1 symbolic ) "default"  )
(   ( M3_M2 symbolic  ) "default"  )
(   ( ND1M2_M1 symbolic  ) "NDrule1"  )
(   ( ND1M3_M2 symbolic  ) "NDrule1"  )
(   ( ND2M2_M1 symbolic  ) "NDrule2"  )
(   ( ND2M3_M2 symbolic  ) "NDrule2"  )

; end of prViaTypes
```

For more information about the `prViaTypes` subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual.*
Specifying Vias That Can Be Stacked: prStackVias()

The prStackVias subclass (section) of the Place and Route Rules class defines the via layers that can be stacked in Preview Gate Ensemble software and Preview Silicon Ensemble software.

**Stack Vias subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Layer names.** Specify the names of the via layers that can be stacked, separated by spaces. A purpose of drawing is assumed. For each layer you specify, the layer-purpose pair with the purpose drawing must be defined in the Layer Definitions class. Also, each via layer specified must be used in a contact or ruleContact device already specified in the Devices class.

```plaintext
prStackVias(
  ; ( layer... )
  ( via via2 )
  ( ( poly1 drawing ) poly2 )
) ; end of prStackVias
```

For more information about the prStackVias subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual.*
Specifying Master Slice Layers: prMastersliceLayers()

The `prMastersliceLayers` subclass (section) of the Place and Route Rules class defines the master slice layers for use in Preview Gate Ensemble software and Preview Silicon Ensemble software.

**Master Slice Layers subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Layer names.** Specify the names of the master slice layers in order, starting with the layer closest to the substrate and progressing to the layer farthest from the substrate. Separate the layer names by a space. A purpose of drawing is assumed. For each layer you specify, the layer-purpose pair with the purpose `drawing` must be defined in the Layer Definitions class.

```plaintext
prMastersliceLayers(
  ; ( layer...          )
  ( diff poly1         )
    ;
    ;
) ; end of prMastersliceLayers
```

For more information about the `prMastersliceLayers` subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual*.

Specifying Via Rules: prViaRules()

The `prViaRules` subclass (section) of the Place and Route Rules class defines the rules for placing vias using the SROUTE or the Preview PowerRoute options of Preview Gate Ensemble software and Preview Silicon Ensemble software.
**Technology File and Display Resource File User Guide**

**Creating a Technology File: Application-Specific Rules**

**Note:** The place-and-route software evaluates the rules and vias listed in this subclass for use on a first-found, first-used basis. If you define more than one rule for a routing situation, the second rule will not be read.

**Via Rules subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Rule name.** Specify any unique string.
- **Via layer name.** Specify the via or vias, enclosed in parentheses and separated by spaces, that can be used. Each via specified must be a `contact` or `ruleContact` device already defined in the Devices class.

```plaintext
prViaRules(

; ( ruleName viaName ( "viaSP21" (M2_M1)

; layer1 dir1 ( wMin wMax overhang metalOverHang )
metal1 "vertical" (.6 1.8 _NA_ _NA_ )

; layer2 dir2 ( wMin wMax overhang metalOverHang )
metal2 "horizontal" (.6 1.8 _NA_ _NA_ )
)

); end of prViaRules
```

**Note:** Continued in following figure.
The following sample illustrates the use of the `prViaRules` subclass:

```plaintext
prViaRules(
    ; ( ruleName viaName
    "viaSP21" (M2_M1)
    ; layer1 dir1 "vertical" ( wMin wMax overhang metalOverHang )
    metal1 "vertical" (.6 1.8 _NA_ _NA_ )
    ; layer2 dir2 "horizontal" ( wMin wMax overhang metalOverHang )
    metal2 "horizontal" (.6 1.8 _NA_ _NA_ ) )
) ; end of prViaRules
```

This code specifies that the router place the M2_M1 contact in the following situation:
The two layers that need a via are `metal1` and `metal2`.

- Both paths have a width between 0.6 and 1.8 user units.
- The `metal1` path is moving vertically.
- The `metal2` path is moving horizontally.
- The contact extends past the `metal1` and `metal2` paths by 0.6 user units.

For more information about the `prViaRules` subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual*.

### Specifying Generated Via Rules: `prGenViaRules()`

A generated via rule specifies the dimensions of a via the router generates as it routes.

The `prGenViaRules` subclass (section) of the Place and Route Rules class defines the generated via rules.

#### Generated Via Rules subclass enclosure

All subclass data must be specified within the parentheses of the subclass enclosure.

- **Rule name.** Specify any unique string for the rule name.
- **Via layer name.** Specify the via or vias, enclosed in parentheses and separated by spaces, that can be used. Each via specified must be a `contact` or `ruleContact` device already defined in the Devices class.

```plaintext
prGenViaRules(
  ; ( ruleName layer
    ( viagen21 via
      ; ( width length xPitch yPitch resistance )
      ( .6  .6  1.2  1.2  _NA_ )
    ; layer1 dir1 ( wMin wMax overhang metalOverHang )
    metal1 "horizontal" ( .6  2.0  .6_  .6 )
    ; layer2 dir2 ( wMin wMax overhang metalOverHang )
    metal2 "vertical" ( .6  2.0  .6  .6 )
    ;
    ;
    ) ;end prGenViaRules
```

**Note:** Continued in following figure.
Creating a Technology File: Application-Specific Rules

Width. Specify the width of the via layer.

Length. Specify the length of the via layer.

x pitch. Specify the x pitch of the via layer.

y pitch. Specify the y pitch of the via layer.

Resistance. Specify the resistance of the via layer.

```
prGenViaRules(
    ; ( ruleName layer ( viagen21 via
        ; ( width length xPitch yPitch resistance )
        ( .6 .6 1.2 1.2 _NA_ )

    ; layer1 dir1 ( wMin wMax overhang metalOverHang )
    metal1 "horizontal" ( .6 2.0 .6_ .6 )

    ; layer2 dir2 ( wMin wMax overhang metalOverHang )
    metal2 "vertical" ( .6 2.0 .6  .6 )
    .
    .
) ;end prGenViaRules
```

Note: Continued in following figure.
Layer name. Specify the name of the routing layer for the top of the via. A purpose of drawing is assumed. The layer-purpose pair with the purpose drawing must be defined in the Layer Definitions class.

Direction. Specify the routing direction for the top routing layer (horizontal or vertical).

Minimum width. Specify the minimum allowable wire width, in user units.

Maximum width. Specify the maximum allowable wire width, in user units.

Overhang. Specify the minimum spacing between the contact cut and the outer edge of the via.

Metal overhang. Specify the minimum overhang of the via to the wire.

Layer 2 specifications. Specify the same set of parameters for the bottom routing layer as for the top routing layer.

The following sample illustrates the use of the `prGenViaRules` subclass:

```plaintext
prGenViaRules(
  ; ( ruleName layer
    ( viagen21 via
      ; ( width length xPitch yPitch resistance )
      ( .6 .6 1.2 1.2 _NA_ )
      ; layer1 dir1 ( wMin wMax overhang metalOverhang )
      metal1 "horizontal" ( .6 2.0 .6_ .6 )
      ; layer2 dir2 ( wMin wMax overhang metalOverhang )
      metal2 "vertical" ( .6 2.0 .6  .6 )
    )
  )
); end of prGenViaRules
```

prViaRules(
    ...
)
This code specifies that the router generate a via with the following dimensions:

- The width and length of a via cut is 0.6
- Via geometries are distributed in an array of 1.2 x and y pitch
- The array expands to fit the path width of the metal1 and metal2 layers (which is between 0.6 and 20 user units)
- The contact extends past the metal1 and metal2 paths by 0.6 user units
- The contact extends past the via cuts by 0.6 user units

For more information about the prGenViaRules subclass, refer to the Technology File and Display Resource File SKILL Reference Manual.
Specifying Turn Via Rules: prTurnViaRules()

When you route a design with Preview Silicon Ensemble software or Preview Gate Ensemble software, the corners of right-angle turns on your routing layer can sometimes be missing. To indicate how you want these gaps handled, you need to specify a turn via rule that fills the gap with a single layer via.

The prTurnViaRules subclass (section) of the Place and Route Rules class defines the rules for placing turn vias during place and route.

**Turn Via Rules subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Rule name.** Specify any unique string for the rule name.
- **Layer name.** Specify the name of the layer to which to apply the rule. A purpose of drawing is assumed. For each layer you specify, the layer-purpose pair with the purpose drawing must be defined in the Layer Definitions class. Specifying the rule forces the router to place a turn via as defined when the routing layer makes a 90-degree turn.
- **Direction list.** Specify the routing directions, in parentheses and separated by a space, to which the rule applies. Usually, horizontal and vertical are specified, in any order.

```
prTurnViaRules()

; ( ruleName  layer  ( directionList ) )
( "TURN1"  metal1  ("vertical"  "horizontal" ) )
( "TURN2"  metal2  ("vertical"  "horizontal" ) )
( "TURN3"  metal3  ("vertical"  "horizontal" ) )

); end of prTurnViaRules
```

For more information about the prTurnViaRules subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual.*
Specifying Nondefault Rules: prNonDefaultRules()

Nondefault rules control Preview Silicon Ensemble software and Preview Gate Ensemble software net routing during optional wide-wire routing. You list wires and vias to be used during wide-wire routing and assign your nondefault rule to a net in your design.

The prNonDefaultRules subclass (section) of the Place and Route Rules class defines the nondefault rules.

**Nondefault Rules subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Rule name.** Specify any unique string for the rule name.

```plaintext
prNonDefaultRules(
    ; ( ruleName
        ( "NDrule1"

    ; ( ( layer width spacing [notch] )... )
        ( metal1 3.8 1.5 1.3 )
        ( metal2 4.8 1.6 _NA_ )
        ( metal3 4.8 1.6 _NA_ )

    ; ( ( viaName [resistance] ) ... )
        ( ND1M2_M1 .8) (ND1M3_M2 .8)

    ; [ ( ( layer1 layer2 minSpace stack ) ... ) ]
        ( metal1 metal1 1.3 nil )
        ( via via2 4.5 nil )

); end of prNonDefaultRules
```

*Note: Continued in following figures.*
Routing layer definitions. Specify each routing layer definition as a set, enclosed in parentheses with the parameters separated by spaces. All of the layer definitions must be within another set of delimiting parentheses. Specify the following parameters for each routing layer:

- **Layer name.** Specify the name of the layer to which to apply the rule. A purpose of drawing is assumed. For each layer you specify, the layer-purpose pair with the purpose drawing must be defined in the Layer Definitions class.

- **Layer width.** Specify the width of the layer, in user units

- **Minimum spacing.** Specify the minimum spacing allowed for the layer, in user units

- **Notch spacing.** Specify the notch spacing allowed for the layer, in user units

```plaintext
prNonDefaultRules(
    ; ( ruleName
      ( "NDrule1"
        ; ( layer width spacing [notch] )... )
      ( ( metal1 3.8 1.5 1.3 )
        ( metal2 4.8 1.6 _NA_ )
        ( metal3 4.8 1.6 _NA_ )
      )

    ; ( ( viaName [resistance] ) ... )
    ( ( ND1M2_M1 .8) (ND1M3_M2 .8) )

    ; [ ( ( layer1 layer2 minSpace stack ) ... ) ]
    ( ( metall metall 1.3 nil )
      ( via via2 4.5 nil )
    )
)
);
```

**Note:** Continued in following figures.
Via device definitions. Specify each via device definition as a set, enclosed in parentheses with the parameters separated by spaces. All of the via device definitions must be within another set of delimiting parentheses. Specify the following parameters for each via device definition:

**Via name.** Specify the name of the via device to which to apply the rule. Each device you specify must be listed in the Devices class.

**Resistance.** Specify the resistance of the via device, in user units.

```lisp
(prNonDefaultRules(
  ; ( ruleName
    ( "NDrule1"

    ; ( ( layer width spacing [notch] ) ... )
    ( ( metal1 3.8 1.5 1.3 )
      ( metal2 4.8 1.6 _NA_ )
      ( metal3 4.8 1.6 _NA_ )
    )

    ; ( ( viaName [resistance] ) ... )
    ( ( ND1M2_M1 .8) (ND1M3_M2 .8) )

    ; [ ( ( layer1 layer2 minSpace stack ) ... ) ]
    ( ( metal1 metal1 1.3 nil )
      ( via via2 4.5 nil )
    )
)
); end of prNonDefaultRules
```

**Note:** Continued in following figure.
**Via layer definitions.** Specify each via layer definition as a set, enclosed in parentheses with the parameters separated by spaces. All of the via layer definitions must be within another set of delimiting parentheses. Specify the following parameters for each via layer:

- **Layer 1 name.** Specify the name, the number, or the layer name and purpose (enclosed in parentheses and separated by a space) of the top routing layer of the via.
- **Layer 2 name.** Specify the name, the number, or the layer name and purpose (enclosed in parentheses and separated by a space) of the bottom routing layer of the via.
- **Minimum spacing.** Specify the minimum spacing allowed between the via layers, in user units.
- **Stack.** Specify whether the via layers can be stacked: `t`=stackable; `nil`=not stackable.

```plaintext
prNonDefaultRules(
  ; ( ruleName
    ( "NDrule1"
    
    ; ( ( layer width spacing [notch] )... )
    ( ( metal1 3.8 1.5 1.3 )
    ( metal2 4.8 1.6 _NA_ )
    ( metal3 4.8 1.6 _NA_ ) )

    ; ( ( viaName [resistance] ) ... )
    ( ( ND1M2_M1 .8) (ND1M3_M2 .8) )

    ; ( [ ( layer1 layer2 minSpace stack ) ... ]
    ( ( metal1 metal1 1.3 nil )
    ( via via2 4.5 nil ) )

  ) ; end of prNonDefaultRules
)
```

For more information about the `prNonDefaultRules` subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual.*
Setting Routing Pitch: prRoutingPitch()

Preview Silicon Ensemble software requires that you specify routing pitch to control its routing. Routing pitch is the minimum allowable spacing, center-to-center, between two geometries on different nets.

The `prRoutingPitch` subclass (section) of the Place and Route Rules class defines the routing pitch for layers that the place-and-route tools use for routing.

**Routing Pitch subclass enclosure.** All subclass data must be specified within the parentheses of the subclass enclosure.

- **Layer name.** Specify the name of the routing layer. A purpose of `drawing` is assumed. For each layer you specify, the layer-purpose pair with the purpose `drawing` must be defined in the Layer Definitions class.

- **Routing pitch.** Specify the routing pitch, in user units, for the routing grid of the layer. Each layer name and routing pitch must be enclosed in parentheses.

```plaintext
prRoutingPitch(
    ; ( layer   pitch  )
    ( metal1  2.4  )
    ( metal2  2.4  )
    ;
) ; end of prRoutingPitch
```

For more information about the `prRoutingPitch` subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual*.
Setting Routing Offset: prRoutingOffset()

Preview Silicon Ensemble software requires that you specify routing offset to control its routing. Routing offset is the distance between the placement grid and the routing grid when there is a routing grid between two placement grids.

The prRoutingOffset subclass (section) of the Place and Route Rules class defines the routing offset of layers the place-and-route tools use for routing.

Routing Offset subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

Layer name. Specify the name of the routing layer. A purpose of drawing is assumed. For each layer you specify, the layer-purpose pair with the purpose drawing must be defined in the Layer Definitions class.

Routing offset. Specify the routing offset, in user units, for the routing grid of the layer. Each layer name and routing offset must be enclosed in parentheses.

```
prRoutingOffset(
    ; ( layer offset )
    ( metal1 0.0 )
    ( metal2 1.2 )
    ...
) ; end prRoutingOffset
```

For more information about the prRoutingOffset subclass, refer to the Technology File and Display Resource File SKILL Reference Manual.
Specifying Overlap Layers: prOverlapLayer()

Overlap layers indicate where cells cannot overlap. This information is optional for Preview Silicon Ensemble software.

The prOverlapLayer subclass (section) of the Place and Route Rules class defines the overlap layers the place-and-route tools use for routing.

Overlap layer subclass enclosure. All subclass data must be specified within the parentheses of the subclass enclosure.

- **Layer name.** Specify the name or names of the routing layer or layers used to display the overlap boundary that indicates where cells cannot overlap. A purpose of boundary is assumed. For each layer you specify, the layer-purpose pair with the purpose boundary must be defined in the Layer Definitions class.

```plaintext
prOverlapLayer(
    ; ( layer...
    ( overlap overlap2 ).
    ) ;end prOverlapLayer
```

For more information about the prOverlapLayer subclass, refer to the *Technology File and Display Resource File SKILL Reference Manual*. 
Creating a Display Resource File

This chapter discusses the following:

- “Methods of Initial Display Resource File Creation” on page 143
- “Display Resource File Contents” on page 143
- “Specifying Display Resources” on page 145
Methods of Initial Display Resource File Creation

You can create a new display resource file by any of the following methods:

■ In a text editor, create a display resource file from scratch
■ Copy a sample display resource file from the Cadence® installation and edit it in a text editor to produce your own display resource file
■ Copy an existing display resource file from your company’s files and edit it in a text editor to produce your own display resource file
■ Dump the display resource data from virtual memory to a new display resource file
■ Edit display resource data in virtual memory with the Display Resource Editor and save the edited data to a display resource file

Whatever method you use, the structure of and requirements for specifying display resources in a display resource file remain the same. This chapter defines how to specify display resources.

Display Resource File Contents

This section defines what a display resource file defines and presents an abbreviated sample display resource file annotated to define the function of each section of the file.

What a Display Resource File Defines

A display resource file defines the following:

■ The different display devices that you use, such as monitors and plotters
■ The display packets that the Cadence design framework II (DFII) software uses to display the layers in your design
■ The colors used in display packets
■ The stipple patterns used in display packets
■ The line styles used in display packets
■ Alias names for display packets
Sample Display Resource File

The following sample illustrates the sections of this file, along with the display resources they define. For a complete sample display resource file, refer to Appendix D, “Technology File and Display Resource File Examples.” For more information about the sections of the display resource file, refer to the Technology File and Display Resource File SKILL Reference Manual.

```plaintext
drDefineDisplay(
    ;( displayName )
    ( display )
    ( display2 )
    ...
    ...
); end of drDefineDisplay

Define Display section. Lists the names of the display devices for which display information is defined in this file.

drDefineColor(
    ;( DisplayName ColorName Red Green Blue Blink )
    ( display white 255 255 255 )
    ( display whiteB 255 255 255 t )
    ( display silver 217 230 255 )
    ...
    ...
); end of drDefineColor

Define Color section. Defines the colors used with various display devices.

drDefineStipple(
    ;( DisplayName StippleName Bitmap )
    ( display blank ( (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0) (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0) (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0) )
    ...
    ...
); end of drDefineStipple

Define Stipple section. Defines the stipple patterns used with various display devices.

drDefineLineStyle(
    ;( DisplayName LineStyle Size Pattern )
    ( display solid 1 ( 1 1 1 ) )
    ( display dashed 1 ( 1 1 1 1 0 0 ) )
    ( display dots 1 ( 1 0 0 ) )
    ...
    ...
); end of drDefineLineStyle

Define Line Style section. Defines the line styles used with various display devices.
```
Specifying Display Resources

This section provides detailed information about specifying display resources in a display resource file.

```
drDefinePacket(
    ;( Display Packet Stip Line Fill Outline [FillStyle])
    ( display blacksolid_S solid solid black black solid )
    ( display blue blank solid blue blue )
    ( display bluedashed_L blank dashed blue blue )
    .
    .
    .
) ; end of drDefinePacket

drDefinePacketAlias(
    ;( displayName packetAlias packetName )
    ( display blsol1 blacksolid_S )
    ( display blue2 blue )
    ( display b3 blue )
    .
    .
    .
) ; end of drDefinePacketAlias
```
Specifying Display Devices: drDefineDisplay()

The drDefineDisplay section of the display resource file lists the display devices for which display resources are defined.

**Define Display enclosure.** Specify all display names within the parentheses of the enclosure.

Display name. List the unique name of each display device, in parentheses, for which you are defining display resources.

```plaintext
drDefineDisplay(
; (displayName )
( display1 )
( display2 )
( plotter1 )
( plotter2 )
.
).
) ; end drDefineDisplay
```


Commonly Used Display Devices

The following table lists commonly used display devices.

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>Color monitor</td>
</tr>
<tr>
<td>hp6</td>
<td>Hewlett-Packard 6-carousel pen plotters</td>
</tr>
<tr>
<td>hp8</td>
<td>Hewlett-Packard 8-carousel pen plotters</td>
</tr>
<tr>
<td>psb</td>
<td>PostScript black-and-white plotters</td>
</tr>
<tr>
<td>versatecb</td>
<td>Versatec and CalComp black-and-white plotters</td>
</tr>
<tr>
<td>versatecc</td>
<td>Versatec and CalComp color plotters</td>
</tr>
<tr>
<td>XBlackWhite</td>
<td>Black-and-white X Window System monitors</td>
</tr>
<tr>
<td>X4PlaneColor</td>
<td>4-plane color X Window System monitors</td>
</tr>
</tbody>
</table>

For more information about setting up plotters, refer to the Plotter Configuration User Guide.
Specifying Colors: drDefineColor()

The drDefineColor section of the display resource file defines the colors to be used in your displays.

**Define Color enclosure.** Specify all color definitions within the parentheses of the enclosure. Specify each complete color definition in parentheses.

- **Display name.** Specify a display name. The display name must be listed in the drDefineDisplay section.
- **Color name.** Specify any unique string as a color name.
- **RGB color values.** Specify the red index, the green index, and the blue index for the color, separated by spaces. Each value must be an integer between 0 and 255, inclusive.
- **Blink.** Optionally specify blinking. \( t = \) blinking; \( \text{nil} = \) not blinking (default).

```
| drDefineColor ( |
| ; ( DisplayName ColorName Red Green Blue [ Blink ] ) |
| ( display white 255 255 255 ) |
| ( display whiteB 255 255 255 t ) |
| ( display silver 217 230 255 ) |
| . |
| ) ; end of drDefineColor |
```

For more information about drDefineColor, refer to the Technology File and Display Resource File SKILL Reference Manual.
Specifying Stipple Patterns: drDefineStipple()

The drDefineStipple section of the display resource file defines the stipple patterns to be used in your displays.

**Define Stipple enclosure.** Specify all stipple definitions within the parentheses of the enclosure. Specify each complete stipple definition in parentheses.

- **Display name.** Specify a display name. The display name must be listed in the drDefineDisplay section.
- **Stipple name.** Specify any unique string as a stipple name.
- **Stipple pattern (bit map).** Specify 1 or t to indicate where the pattern is solid and 0 or nil to indicate where the pattern is blank.

```

drDefineStipple(
; ( displayName StippleName Stipple Pattern)
( display checker
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
  (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
  (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
  (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
  (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
  (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
  (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
  (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
  (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1))
); end of drDefineStipple)
```

For more information about drDefineStipple, refer to the Technology File and Display Resource File SKILL Reference Manual.
Specifying Line Styles: drDefineLineStyle()

The drDefineLineStyle section of the display resource file defines the line styles to be used in your displays.

**Define Line Style enclosure.** Specify all line style definitions within the parentheses of the enclosure. Specify each complete line style definition in parentheses.

| Display name. Specify a display name. The display name must be listed in the drDefineDisplay section. |
| Line style name. Specify any unique string as a line style name. |
| Line size. Specify the thickness of the line pattern, in pixels. |
| Line pattern. Specify 1 or t to indicate where the line is solid and 0 or nil to indicate where the line is blank. |

```plaintext
drDefineLineStyle(
    ;( DisplayName   LineStyle     Size    Pattern )
    ( display       solid 1 (1 1 1) )
    ( display       dashed 1 (1 1 1 0 0) )
    ( display       dots 1 (1 0 0) )
    ( display       dashDot 1 (1 1 0 0 1 0 0) )
    ( display       shortDash 1 (1 1 0 0) )
    ( display       doubleDash 1 (1 1 1 0 0 1 1 0 0) )
    ( display       hidden 1 (1 0 0 0) )
    ( display       thickLine 3 (1 1 1) )
    ( display       bigDash 2 (1 1 1 0 0 1 1 1 0 0) )
)
```

For more information about drDefineLineStyle, refer to the Technology File and Display Resource File SKILL Reference Manual.
Specifying Display Packets: drDefinePacket()

The drDefinePacket section of the display resource file defines the display packets to be used for your displays.

Define Display Packet enclosure. Specify all display packet definitions within the parentheses of the enclosure. Specify each complete definition in parentheses.

- **Display name.** Specify a display name. The display name must be listed in the drDefineDisplay section.
- **Packet name.** Specify any string unique to the display as a display packet name. For guidelines, refer to “Packet Naming Conventions” on page 151.
- **Stipple pattern.** Specify a stipple pattern name. The name specified must be for a stipple pattern defined in the drDefineStipple section.
- **Line style.** Specify a line style name. The name specified must be for a line style defined in the drDefineLineStyle section.
- **Fill color.** Specify a fill color name. The name specified must be for a color defined in the drDefineColor section.
- **Outline color.** Specify an outline color name. The name specified must be for a color defined in the drDefineColor section.
- **Fill style.** Optionally specify a fill style name. The name must be for a fill style defined in “Fill Styles” on page 153. Overrides stipple and line styles.

```

drDefinePacket()

;( Display Packet Stipple Line Fill Outline [FillSt])

( display blacksolid_S solid solid black black solid )
( display blue blank solid blue blue      )
( display bluedashed_L blank dashed blue blue x )
( display bluevZigZag_S ZigZag solid blue blue )
( display browndashed_L blank dashed brown brown )

); end of drDefinePacket)
```

Packet Naming Conventions

It is strongly recommended that you adhere to the packet naming conventions outlined in this section when you name a display packet.

A packet name has four sections:

- Color (fill color and/or outline color)
- Stipple
- Line style
- SLNB extension

The name has the following structure. Note that the packet name must not contain spaces, and the underscore character (_) must precede the SLNB extension.

\[\text{color}[\text{stipple}][\text{line}][\_][S][L][N][B]\]

where:

- \text{color} specifies the fill color and/or the outline color in the following format:

  \[\text{[fill]} [\text{outline}]\]

  At least one color name must be specified. Specify colors according to the following rules:

  - When the fill and outline colors are different, specify the fill color followed by the outline color.
  
  - When the fill and outline colors are the same, specify the one color.
  
  - When the fill and outline colors are the same, but the outline color is blinking (as defined with \text{drDefineColor}), specify the one color when the outline color name is the same as the fill color name with a B appended (for example, \text{white} and \text{whiteB}). If the outline color name is constructed in any other way (for example, \text{white2}), specify both colors (\text{white}white2).

- \text{stipple} specifies a stipple pattern name. This field is optional; no entry indicates the default stipple pattern name, \text{blank}. 
Creating a Display Resource File

The following table shows sample combinations of color, stipple, and line resources and the packet names correctly built according to the packet naming conventions.

<table>
<thead>
<tr>
<th>Fill Color</th>
<th>Outline Color</th>
<th>Stipple Pattern</th>
<th>Line Style</th>
<th>Packet Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>blue</td>
<td>blue</td>
<td>blank</td>
<td>solid</td>
<td>blue</td>
</tr>
<tr>
<td>blue</td>
<td>blue</td>
<td>blank</td>
<td>dashed</td>
<td>bluedashed_L</td>
</tr>
<tr>
<td>blue</td>
<td>blue</td>
<td>solid</td>
<td>solid</td>
<td>bluesolid_S</td>
</tr>
<tr>
<td>blue</td>
<td>blue</td>
<td>metal1S</td>
<td>solid</td>
<td>bluemetal1S_S</td>
</tr>
<tr>
<td>blue</td>
<td>blue</td>
<td>metal1S</td>
<td>none</td>
<td>bluemetal1S_SN</td>
</tr>
<tr>
<td>cream</td>
<td>white</td>
<td>comp</td>
<td>solid</td>
<td>creamwhitecomp_S</td>
</tr>
<tr>
<td>green</td>
<td>green</td>
<td>brick</td>
<td>mLine</td>
<td>greenbrickmLine</td>
</tr>
<tr>
<td>red</td>
<td>red</td>
<td>X</td>
<td>thickLine</td>
<td>redXthickLine</td>
</tr>
<tr>
<td>red</td>
<td>blinking red</td>
<td>x</td>
<td>solid</td>
<td>redSB</td>
</tr>
<tr>
<td>red</td>
<td>blinking red</td>
<td>solid</td>
<td>none</td>
<td>redred2solid_SNB</td>
</tr>
</tbody>
</table>
**Fill Styles**

The following table defines the fill styles.

<table>
<thead>
<tr>
<th>Fill Style</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>outline</td>
<td></td>
</tr>
<tr>
<td>solid</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>stipple</td>
<td></td>
</tr>
</tbody>
</table>

(The stipple pattern specified determines the fill.)

<table>
<thead>
<tr>
<th>Fill Style</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>outline stipple</td>
<td></td>
</tr>
</tbody>
</table>

(The stipple pattern specified determines the fill.)

**Note:** When you specify a fill style, it overrides the stipple and line styles.

**Customizing Display Packets for Plotting**

You can set up your display resource file so that a display packet appears differently on different display devices. For example, if your plotter uses only 7 colors and you display your design using 12 colors, you can modify the display packets used by the layers in your design specifically for the plotting device. As shown in the following figure, you define a display packet with the same name but different content for each display device.
Specifying Display Packet Aliases: drDefinePacketAlias()

The `drDefinePacketAlias` section of the display resource file defines alias names for the display packets to be used for your displays. You can then use the alias name as well as the display packet name to access the same display packet. This function allows flexibility in assigning display packets to display devices. You can alias an existing display packet name to another display packet name to change the display packet in use for a given display device.

**Define Display Packet Alias enclosure.** All alias definitions must be specified within the parentheses of the enclosure. Specify each alias definition in parentheses.

<table>
<thead>
<tr>
<th>Display name.</th>
<th>Specify a display name. The display name must be listed in the <code>drDefineDisplay</code> section.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias name.</td>
<td>Specify any unique string as an alias for the display packet name.</td>
</tr>
<tr>
<td>Display packet name.</td>
<td>Specify the name of the display packet to which you want to assign the alias. The name must be for a display packet defined in the <code>drDefinePacket</code> section.</td>
</tr>
</tbody>
</table>

```

drDefinePacketAlias

( displayName packetAlias packetName )
( display   blosoll                  blacksolid_S   )
( display   blue2                    blue          )
( display   b3                        blue          )

) ; end of drDefinePacketAlias
```

For more information about `drDefinePacketAlias`, refer to the *Technology File and Display Resource File SKILL Reference Manual*. 
Preparing Files for Use with a Design

This chapter discusses the following:

- “Generating a Technology Library” on page 156
- “Checking a Technology File for Conformance to Application Requirements” on page 159
- “Attaching a Technology Library to a Design” on page 161
- “Detaching a Technology Library from a Cell or Cellview” on page 166
- “Ensuring Desired Display Resource File Usage” on page 167
Generating a Technology Library

Compiling an ASCII Technology File into a Library

If you have an ASCII technology file that you want to use to create a technology library, do the following:

1. From the CIW menu banner, choose Technology File – New.
   The New Technology Library form appears.

2. In the Technology Library Name field, type the name of the new technology library you want to create.
3. Click Load ASCII Technology File.
4. Type the name of the ASCII technology file to compile.
5. Choose the directory in which to create the new Cadence design framework II (DFII) library.

For a description of this form, see Appendix A.
Technology File and Display Resource File User Guide
Preparing Files for Use with a Design

- To descend into the directory hierarchy, double-click a directory.
- To ascend the directory hierarchy, double-click the “..” directory.
- To choose a directory, click the directory name.

6. If you want to use a design management system, choose the design manager from the Design Manager cyclic field.
   - If your design manager environment is set to prompt you to check-in all or views, the system prompts you to check-in the technology file.
   - If your design manager environment is set to check-in files or none, messages in the CIW tell you that check-in of the technology library files and cellviews is skipped because of preferences.

For more information about setting check-in options, see “Setting Automatic Checkout and Checkin Preferences” in the Design Framework II Help.

For information about the Cadence® team design manager (TDM), refer to the Team Design Manager User Guide. For information about working with another design manager with Cadence tools, refer to the Cadence Application Infrastructure User Guide.

7. Click OK.

The system compiles the ASCII file and creates the new DFII technology library. The directory of the new library contains a binary version of the ASCII file called techfile.cds and cellviews of the devices defined in the technology file.

Note: A DFII library can contain only one binary technology file. The name of the binary file must be techfile.cds. Do not change the name of the file.

To use the new technology library, you must attach it to a DFII design or design library. For more information, refer to “Attaching a Technology Library to a Design” on page 161.

Creating a New Technology Library from an Existing Technology Library

You can create a new technology library by loading the technology file from an existing technology library.

1. From the CIW menu banner, choose Technology File – New.

   The New Technology Library form appears.
2. In the Technology Library Name field, type the name of the new technology library to create.

3. Click Load Existing Technology Library.

4. Choose the technology library you want to load from the cyclic field.

   For a description of this form, see Appendix A.

5. Choose the directory in which you want to create the new DFII library.

   - To descend the directory hierarchy, double-click a directory name.
   - To ascend the directory hierarchy, double-click the “..” directory.
   - To choose a directory, click the directory name.

6. If you want to use a design management system, choose the design manager from the Design Manager cyclic field.

   - If your design manager environment is set to prompt you to check-in all or views, the system prompts you to check-in the technology file.
If your design manager environment is set to check-in *files or none*, messages in the CIW tell you that check-in of the technology library files and cellviews is skipped because of preferences.

For more information about setting check-in options, see “Setting Automatic Checkout and Checkin Preferences” in the *Design Framework II Help*.

For information about the Cadence team design manager (TDM), refer to the *Team Design Manager User Guide*. For information about working with another design manager with Cadence tools, refer to the *Cadence Application Infrastructure User Guide*.

7. Click *OK*.

   The system copies the binary technology file `techfile.cds` and the devices from the existing library to the new DFII library.

   **Note:** The name of the binary file must be `techfile.cds`. Do not change the name of the file. A DFII library can contain only one binary technology file.

To use the new technology library, you must attach it to a DFII design or design library. Refer to “Attaching a Technology Library to a Design” on page 161 for more information.

**Checking a Technology File for Conformance to Application Requirements**

After you have compiled your ASCII technology file, you can check your file for use with the DFII applications.

1. From the CIW menu banner, choose *Technology File – Check*.

   The Check Technology File form appears.
For a description of this form, see Appendix A.

2. From the Technology Library cyclic field, choose the library containing the technology file to check.

3. From the Application radio buttons, choose the applications you plan to use.

4. From the Print Message Types radio buttons, choose the types of messages you want to see.

5. Click Apply or OK.

The software checks the technology file for conformance to the requirements of the applications you selected. Messages appear in the CIW and in the CDS.log file in your home directory. The following are sample error messages:

*ERROR* Type 53: ("Group" "drawing") and ("Group" "label") LPPs are missing
*ERROR* Type 91: "bigM1_pin" is defined as maskable. It should be defined as non-maskable.
*ERROR* Type 93: "bigM1_pin" is not a valid preView/P&R softpin since "metal1" is not a prRouting layer.

Correct the technology data so that it conforms to the requirements of the applications, regenerate the technology library, and recheck your data for conformance to application requirements.
Attaching a Technology Library to a Design

You must attach a technology library to a design or design library to use it in your design process. You can attach any technology library to a design library, cell, or cellview. Each cell or cellview in a design library uses the technology library attached to the design library unless another technology library is specifically attached to the cell or cellview.

When you attach a technology library to a design library, cell, or cellview,

- The software updates the technology devices placed in each cellview to reference the new technology library
- The software adds the `techLibName` property with the new technology library name to the library, cell, or cellview.

  If the property already exists, the software updates it

  The `techLibName` property identifies the technology library attached to the library, cell, or cellview.

If you want to delete the attachment of a cell or cellview and return it to the default of using the design library attachment, you must detach the cell by doing the following:

- Reattach the technology library used by the design library to the cell or cellview (to update the technology devices)
- Delete the `techLibName` property from the cell or cellview

In the following example, the design library `designLib` is attached to `techLib1`. The cell `cellA` is attached to `techLib2`. All of the views in `cellB` use `techLib1`. All of the views in `cellA` use `techLib2`.

If you attach `designLib` to a new technology library, the software updates `cellB` and all of its views to use the new technology library, but `cellA` and all of its views continue to use `techLib2`.

```
techLib1  techLibName = techLib1  designLib
  
  techLib2  techLibName = techLib2  cellA
    
    layout  layout1
  
  cellB
    
    layout  layout1
```
Attaching a Technology Library to a Design Library

To attach a technology library to a design library, you must have write permission for the design library property file and all unattached cellviews.

1. From the CIW menu banner, choose Technology File – Attach To.
   
   The Attach Technology Library to Design Library form appears.

   ![Attach Technology Library to Design Library form]

   - Choose the design library.
   - Click here to choose the library from the Library Browser.
   - Choose the library containing the technology file.
   - For a description of this form, see Appendix A.

2. From the Technology Library cyclic field, choose the library containing the technology file you want to assign to the design library.

3. Click OK.

   The system updates the properties of the design library and updates the technology devices in each unattached cellview to reference the new technology file.

   If you use a design management system, all files and cellviews in the library must be checked out or your environment must be set to automatically check-out all properties and cellviews.

   For more information about setting check-out options, see “Setting Automatic Checkout and Checkin Preferences” in the Design Framework II Help.

   Upon completion of the attach command, a message similar to the following appears in your CIW:
Processing (designLib CellA layout) for device update. Changed device reference (techLib3 M1_P symbolic) to (cellTechLib M1_P symbolic). Processing (designLib CellB layout) for device update. Changed device reference (techLib3 M2_M1 symbolic) to (cellTechLib M2_M1 symbolic). Design library ‘designLib’ successfully attached to technology library ‘cellTechLib’.

Attaching a Technology Library to a Design Cell or Cellview

If you have a cell or cellview that requires different technology data than that contained in the technology library assigned to the design library, attach another technology file to the cell or cellview.

1. From the CIW menu banner, choose Technology File – Attach To.

   The Attach Technology Library to Design Library form appears.

   ![Diagram of Attach Technology Library to Design Library form]

   For a description of this form, see Appendix A.

2. Choose the cell or cellview to attach by doing one of the following:

   - To attach the same technology library to all views of a cell, click Cell and type the cell name in the adjacent field.
   - To attach the technology library to one cellview, click Cell and View and type the cell and cellview names in the adjacent fields.
   - To choose the library, cell, or cellview from the Library Browser, click the Browse button.
3. From the Technology Library cyclic field, choose the technology library you want to attach to the cell or cellview.

4. In the Attach Technology Library to Design Library form, click OK.

   The system updates the properties of the cell or cellview you specified and updates the technology devices in each cellview to reference the new technology file.

   If you use a design management system, the cell or cellview must be checked out or your environment must be set to automatically check-out cellviews.

   For more information about setting check-out options, see “Setting Automatic Checkout and Checkin Preferences” in the Design Framework II Help.

   Upon completion of the attach command, a message similar to the following appears in your CIW:

   Changed device reference (techLib1 pTran symbolic) to (cellTechLib pTran symbolic).
   Changed device reference (techLib1 nTran symbolic) to (cellTechLib nTran symbolic).
   Design cellview ('tutorial' 'nand2' 'layout') successfully attached to technology library 'cellTechLib'.
Device Definitions Used by the Software When You Attach One Technology File After Another to a Design Library

The following illustrates what happens with device definitions when you attach one technology file after another to your design library:

1. The design library is using Technology Library A.

2. You attach Technology Library B to the design library.
   The design library then uses the following:
   - From Library B: m1_m2, m1_m4, m1_m5, m1_m6
   - From Library A: m1_m3, m1_m7

3. You attach Technology Library C to the design library.
   The design library then uses the following:
   - From Library C: m1_m2, m1_m4, m1_m5
   - From Library B: m1_m6
   - From Library A: m1_m3, m1_m7

<table>
<thead>
<tr>
<th>Technology Library A</th>
<th>Technology Library B</th>
<th>Technology Library C</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1_m2</td>
<td>m1_m2</td>
<td>m1_m2</td>
</tr>
<tr>
<td>m1_m3</td>
<td>m1_m4</td>
<td>m1_m3</td>
</tr>
<tr>
<td>m1_m4</td>
<td>m1_m5</td>
<td>m1_m4</td>
</tr>
<tr>
<td>m1_m5</td>
<td>m1_m6</td>
<td>m1_m5</td>
</tr>
<tr>
<td>m1_m6</td>
<td></td>
<td>m1_m6</td>
</tr>
<tr>
<td>m1_m7</td>
<td></td>
<td>m1_m7</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Note: After you attach Library B, the design software uses the device definition in Library A for any device not also in Library B (m1_m3, m1_m7). When you attach Library C, the design software still points to Library A for those devices, even though they are also in Library C.

If you have already performed the sequence outlined in the description, but want to use only the devices in a new technology library (for example, Technology Library C above) rather than any of those in the original library, do the following:

1. Reattach your design library to the original technology file (for example, Technology Library A).

2. Attach your design library to the new technology file (for example, Technology Library C).
Detaching a Technology Library from a Cell or Cellview

To delete the attachment of a cell or cellview to a technology library, do the following:

1. To update the devices in the cell or cellview to reference the design library’s technology library, attach the cell or cellview to the technology library currently assigned to the design library.

2. To display the Library Manager from the CIW menu banner, choose Tools – Library Manager.

3. In the Library Manager, click the design library and cell names and, if necessary, the cellview name.

4. From the Library Manager menu, choose Edit – Properties.

   The View Property Editor form or Cell Property Editor form appears.

   If you use a design management system, the cell or cellview properties must be checked out or your environment must be set to automatically check-out properties.

   For more information about setting check-out options, see “Setting Automatic Checkout and Checkin Preferences” in the Design Framework II Help.

5. To choose the property to delete, click techLibName.
6. In the Cell Property Editor form or View Property Editor form, click *Delete*.

7. To close the form, click *OK*.

The software removes the property from the cell or cellview. The unattached cellview will now use the technology library attached to the design library.

**Ensuring Desired Display Resource File Usage**

Keep in mind how the software loads and uses display resource files upon initialization to ensure that you are using display resources in the way you want. For details, refer to “How the Design Framework II Software Handles Multiple Display Resource Files” on page 29.
Editing, Reusing, and Merging Technology File Data

This chapter discusses the following:

- “The Technology File Updating Process” on page 169
- “Methods for Editing a Technology File” on page 170
- “Reusing a Technology File or Library to Build a New Library” on page 170
- “Loading Technology Data into Virtual Memory” on page 174
- “Editing Class Data through the CIW Pull-Down Menus” on page 178
- “Editing Class Data with SKILL Functions” on page 189
- “Checking a Technology File for Conformance to Cadence Application Requirements” on page 201
- “Discarding an Edited Technology File from Virtual Memory (Reloading Data from Disk)” on page 201
- “Saving a Technology File Edited in Virtual Memory to Disk” on page 202
- “About Unsaved Changes Message Boxes” on page 203
The Technology File Updating Process

The following summarizes the general process for updating or creating a new technology file:

- Edit an existing ASCII technology file with a text editor if you want to actually replace the data in that file with new data
  
  or

- Copy an existing ASCII technology file to another file and edit it with a text editor to produce a new technology file
  
  or

- Dump an ASCII technology file from an existing technology library and edit it with a text editor to produce a new technology file
  
  or

- Edit technology data in virtual memory through the CIW pull-down menus or Cadence® SKILL commands and save the edited technology data to the current technology library on disk

  Compile the new ASCII technology file

  Check the technology file for conformance to application requirements

  Attach the technology library to a design, cell, or cellview
Methods for Editing a Technology File

You can edit an existing technology file using the following methods:

- Edit the ASCII file in a text editor and use the Technology File – Load command to compile and load it into virtual memory, either merging it with the technology data already in virtual memory or replacing the technology data already in virtual memory with it.

  For more information about the syntax of the ASCII technology file, refer to chapters 3 through 5 in this manual.

- Edit the technology file in virtual memory through the CIW pull-down commands.

  Note: Not all technology file data is accessible through the CIW pull-down commands. For classes of data that you cannot edit this way, you must use one of the other methods. See “Class Data Accessible through the CIW Pull-Down Menus” on page 178 for details.

- Use SKILL functions to load a binary technology file and update it in memory.

  See “Editing Class Data with SKILL Functions” on page 189 for details. For more information about the technology file SKILL functions, refer to the Technology File and Display Resource File SKILL Reference Manual.

Reusing a Technology File or Library to Build a New Library

This section discusses the following topics:

- Creating an ASCII technology file from a binary technology library
- Copying a technology library to use as a basis for creating a new technology library

Creating an ASCII File from a Binary Technology File

To obtain a writable ASCII technology file, you can dump all or a portion of a technology file from the binary technology library to an ASCII file.

Note: A dumped ASCII technology file differs in the following ways from the original ASCII technology file that was compiled to create the technology library:

- It does not contain comments identified by semicolons (;) in the original ASCII file, although it does contain comments made with the comment statement.
- It contains all of the data from originally included files and no include statements.
It contains any changes made in virtual memory and saved to disk during design sessions.

To create an ASCII technology file from a binary technology library, do the following:

1. From the CIW menu banner, choose Technology File – Dump.

   The Dump Technology File form appears.

   ![Dump Technology File form]

   - Choose the library of the technology file to dump.
   - Click the classes of the technology file you want to dump or click Select All to dump the entire file.
   - Type the path and name of the file you want to create.

   For a description of this form, see Appendix A.

2. From the Technology Library cyclic field, choose the library containing the technology file you want to dump.

3. Click on the classes of the technology file you want to dump.

   Click Select All to dump all classes.

4. In the ASCII Technology File field, type the name of the ASCII file you want to create.

   **Caution**

   Do not overwrite an existing ASCII technology file. Instead, dump your data to a temporary file. While an ASCII file produced with the Technology File – Dump command does contain comments made with the technology file comment function, it does not contain any comments made by preceding them with a semicolon (;) or any of the SKILL programs that an original ASCII file might contain.

5. Click OK.
The ASCII file opens in a shell window. You can edit this file.

For more information about the syntax of the classes and subclasses (sections) of the technology file, refer to chapters 2 through 5 of this user guide and “Technology File ASCII Syntax” in the Technology File and Display Resource File SKILL Reference Manual.

You can dump, edit, and load the ASCII file until you are satisfied with your changes. (For information on loading the technology file and merging its technology data with the technology data already in the technology library in virtual memory, see “Merging New Technology Data into an Existing Technology Library” on page 174. For information on loading the technology file and replacing the technology data in the technology library in virtual memory with the new technology file data, see “Replacing an Existing Technology File in a Technology Library” on page 176.) Then, if necessary, copy the changes in the ASCII file to your annotated “golden” ASCII file.

**Copying a Technology Library to Use As a Basis for Creating a New Technology Library**

You can copy an entire technology library to use as a basis for creating a new technology library. To do so, perform the following steps:

1. From the CIW menu banner, choose *Technology File – New.*

   The New Technology Library form appears.
For a description of this form, see Appendix A.

2. Turn on the Load Existing Technology Library radio button.

3. From the Load Existing Technology Library cyclic field, choose the technology library to copy.

4. In the Technology Library Name field, type the path and name of the technology library to create. (You can navigate your directory structure in the Directory list box.)

5. From the Design Manager cyclic field, choose No DM, if you are not working in a design manager environment, or DM, if you are working in a design manager environment.

6. Click Apply or OK.
The software creates a copy of the selected technology library in the specified directory. You can dump, edit, and recompile the technology file to alter the technology library or you can load the technology library and edit technology data in virtual memory.

**Loading Technology Data into Virtual Memory**

This section explains how to load data into virtual memory in two ways. With the *Technology File – Load* command, you can merge technology data in a technology file with the technology data already in virtual memory or you can replace the technology data in virtual memory with different technology data in a technology file.

**Merging New Technology Data into an Existing Technology Library**

You can define new or edited technology data in an ASCII technology file and then merge that data with an existing technology library by compiling and loading the new technology file according to the following steps (refer to “Replacing an Existing Technology File in a Technology Library” on page 176 for information on replacing the technology library):

1. From the CIW menu banner, choose *Technology File – Load*.

   The Load Technology File form appears.

   ![Load Technology File Form](image)

   - Type the name of the text file you want to load.
   - Choose the classes you want to load.
   - Choose the technology library for the technology file.
   - Choose *Merge*.

   For a description of this form, see Appendix A.

2. In the *ASCII Technology File* field, type the name of the ASCII technology file you want to compile and load.
3. Click the classes you want to compile and load from the ASCII technology file.

   To load all classes, click Select All.

4. From the Technology Library cyclic field, choose the technology library into which you want to compile and load the ASCII technology file data.

5. Click Merge.

   **Note:** When you choose Merge, existing functions in the technology library that are order dependent are replaced. New functions that are order dependent and order independent are appended to the corresponding functions in virtual memory.

6. Click OK.

   The software compiles the technology file and loads it into virtual memory. If your ASCII file does not contain all of the classes you chose, a dialog box appears listing the missing classes. Click OK to continue or Cancel to quit.

   If you use a design management system, the following also occurs:

   ❑ If your environment is set to prompt you to check out all or views, and you have not already checked out the technology file, the system prompts you to check out the technology file for edit.

   ❑ If your environment is set to check out files or none, or you do not have permission to check out the technology file, or someone else already has it checked out, the system loads the ASCII file but does not prompt you to check out the technology file.

   For more information about setting check-out options, see “Setting Automatic Checkout and Checkin Preferences” in the Design Framework II Help.

7. If you are prompted to check out the technology file, choose a check-out option and click OK.

   Choose yes or no.

   ❑ If you choose yes, the system checks out the technology file and loads it into virtual memory with the data compiled from the ASCII file.

   ❑ If you choose no, the system loads the technology file into virtual memory with the data compiled from the ASCII file but does not check the technology library out.
For information about when each class is compiled and when the process is finished, look for messages in the CIW. For example:

Compiling class ‘layerDefinitions’....
Compiling class ‘devices’....
Compiling class ‘physicalRules’....
Compiling class ‘compactorRules’....
Technology file ‘~/tutorial.tf’ loaded successfully.

Replacing an Existing Technology File in a Technology Library

You can define new or edited technology data in an ASCII technology file and then replace an existing technology library by compiling and loading the new technology file according to the following steps (refer to “Merging New Technology Data into an Existing Technology Library” on page 174 for information on merging the technology data with the existing technology library):

1. From the CIW menu banner, choose **Technology File – Load**.

   The Load Technology File form appears.

2. In the ASCII Technology file field, type the name of the ASCII technology file you want to compile and load.

3. Click the classes you want to compile and load from the ASCII technology file.

   To load all classes, click **Select All**.

   For a description of this form, see **Appendix A**.
4. From the *Technology Library* cyclic field, choose the technology library into which you want to compile and load the ASCII technology file data.

5. Click *Replace*.

   **Note:** When you choose *Replace*, the software replaces the entire technology library to reflect the technology data defined in the replacement technology file.

6. Click *OK*.

   The software compiles the technology file and loads it into virtual memory. If your ASCII file does not contain all of the classes you chose, a dialog box appears listing the missing classes. Click *OK* to continue or *Cancel* to quit.

   If you use a design management system, the following also occurs:

   - If your environment is set to prompt you to check out all or views, and you have not already checked out the technology file, the system prompts you to check out the technology file for edit.

   - If your environment is set to check out files or none, or you do not have permission to check out the technology file, or someone else already has it checked out, the system loads the ASCII file but does not prompt you to check out the technology file.

   For more information about setting checkout options, see “Setting Automatic Checkout and Checkin Preferences” in the *Design Framework II Help*.

7. If you are prompted to check out the technology file, choose a check-out option and click *OK*.

   Choose *yes* or *no*.

   - If you choose *yes*, the system checks out the technology file and loads it into virtual memory with the data compiled from the ASCII file.

   - If you choose *no*, the system loads the technology file into virtual memory with the data compiled from the ASCII file but does not check the technology library out.

   For information about when each class is compiled and when the process is finished, look for messages in the CIW. For example:

   - Compiling class ‘layerDefinitions’....
   - Compiling class ‘devices’....
   - Compiling class ‘physicalRules’....
   - Compiling class ‘compactorRules’....
   - Technology file ‘~/tutorial.tf’ loaded successfully.
Editing Class Data through the CIW Pull-Down Menus

You can edit most technology file data in virtual memory when you are running the design software. CIW pull-down menus allow access to and manipulation of technology file data; changes you make in virtual memory become active during your software session, although they do not apply to future sessions unless you save them to a technology library for future use.

This section defines the classes and subclasses (sections) of the technology file you can edit through CIW pull-down menus and points you to the sections in this user guide containing more detailed information on each editing task.

Class Data Accessible through the CIW Pull-Down Menus

The following table summarizes which classes and subclasses (sections) you can edit through the CIW pull-down menus and which you must edit with a text editor:

<table>
<thead>
<tr>
<th>Class</th>
<th>Subclass</th>
<th>CIW Pull-Down Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls</td>
<td>techParams</td>
<td><strong>Technology File – Set Up</strong></td>
</tr>
<tr>
<td></td>
<td>techPermissions</td>
<td>Not accessible; must edit with a text editor</td>
</tr>
<tr>
<td>Layer Definitions</td>
<td>All</td>
<td><strong>Technology File – Edit Layers</strong></td>
</tr>
<tr>
<td>Devices</td>
<td>All</td>
<td>Not accessible; must edit with a text editor</td>
</tr>
<tr>
<td>Layer Rules</td>
<td>All</td>
<td><strong>Technology File – Set Up</strong></td>
</tr>
<tr>
<td>Physical Rules</td>
<td>All</td>
<td><strong>Technology File – Set Up</strong></td>
</tr>
<tr>
<td>Electrical Rules</td>
<td>All</td>
<td><strong>Technology File – Set Up</strong></td>
</tr>
<tr>
<td>Layout Editor Rules</td>
<td>All</td>
<td><strong>Technology File – Set Up</strong></td>
</tr>
<tr>
<td>Virtuoso® XL Rules</td>
<td>All</td>
<td><strong>Technology File – Set Up</strong></td>
</tr>
<tr>
<td>Virtuoso Compactor Rules</td>
<td>All</td>
<td><strong>Technology File – Set Up</strong></td>
</tr>
<tr>
<td>Layout Synthesizer Rules</td>
<td>All</td>
<td><strong>Technology File – Set Up</strong></td>
</tr>
<tr>
<td>Place and Route Rules</td>
<td>All</td>
<td>Not accessible; must edit with a text editor</td>
</tr>
</tbody>
</table>
The CIW Pull-Down Menus

Two pull-down menus on the CIW access forms that allow you to edit class data in your technology file in virtual memory. They are

- Technology File – Edit Layers
- Technology File – Set Up

Technology File – Edit Layers brings up the Layer Purpose Pair Editor form, in which you can edit, delete, and add layer definitions.

For a description of this form, see Appendix A.

For detailed information on editing layer definitions, refer to Chapter 9, “Editing Class Data through CIW Pull-Down Menus: Controls and Layer Definitions.”
Technology File – Set Up brings up the following initial Technology File Set Up form:

![Technology File Set Up Form](image)

For a description of this form, see Appendix A.

On this form, you can choose the technology library and the class data you want to edit.

When you click on a class in the left column (Classes), the form displays the subclasses in the right column (Rules), as shown in the following example.
When you click on a class in the left column and choose File – Edit from the form menu banner or when you double-click on a class in the left column, the Set Up function brings up another form for you to use to begin editing.

To close the Technology File Set Up form, from the menu banner, choose File – Close.

For detailed information on editing the Controls rules, refer to Chapter 9, “Editing Class Data through CIW Pull-Down Menus: Controls and Layer Definitions.”

For detailed information on editing generic rules (layer rules, physical rules, and electrical rules), refer to Chapter 10, “Editing Class Data through CIW Pull-Down Menus: Generic Rules.”

For detailed information on editing application-specific rules (layout editor rules, Virtuoso XL rules, Virtuoso compactor rules, and layout synthesizer rules), refer to Chapter 11, “Editing Class Data through CIW Pull-Down Menus: Application-Specific Rules.”
Layer Browsers

When you choose a class to edit from the Technology File Set Up form, the software displays a form specifically designed for editing that particular data class. Many of these forms contain a *Browse* button that allows you to display a Layer Browser from which you can browse through and choose existing layers from the technology library.

Sample Layer Browser Forms

The following is a sample Layer Browser form for rules that use a single layer:

For a description of this form, see Appendix A.
The following is a sample Layer Browser form for rules that use two layers:

For a description of this form, see Appendix A.
The following is a sample Layer Browser form for the viaLayers Layer Rules subclass:

For a description of this form, see Appendix A.
The following is a sample Layer Browser form for the symWires Compactor Rules subclass:

For a description of this form, see Appendix A.
Example of Using a Layer Browser

When you choose a layer in a Layer Browser, the software automatically loads it into the appropriate field in the current Set Up form. For example, the following is the Set Up form for editing the Virtuoso XL Rules class:

![Example of Using a Layer Browser](image)

When you choose *Browse* on this form, the software displays the Layer Browser. As you click on layers in this Layer Browser, the software highlights them in the browser and automatically
loads the layer names into the appropriate fields on the Set Up form. The following illustrates choosing layers in the Layer Browser:
The following illustrates how the software loads the layers selected in the Layer Browser into the *Layer* fields on the Set Up form:

For details on using the various Set Up forms, see chapters 9, 10, and 11.
Editing Class Data with SKILL Functions

The Controls Class

Adding a Control Parameter

To add a parameter to the techParams subclass (section) of a technology file in virtual memory with SKILL functions, type the techSetParam SKILL function in the CIW command line. If the techParams subclass does not exist, this function creates it. The following is part of an interactive session in the CIW that sets control parameters and uses them to create spacing rules:

```
techSetParam(tf "spacing1" 0.6)
techSetParam(tf "spacing2" 0.3)
techGetParams(tf)
```

```
(("spacing1" 0.6)
 ("spacing2" 0.3))
```

Create a spacing rule using the parameters as the value expression. (This is an example of a minimum spacing rule set for the metal2 and poly1 layers. All objects created on layer-purpose pairs of metal2 and poly1 must be separated by the distance defined by the expression.)

```
techSetSpacingRule(tf "minSpacing" '(techSetParam("spacing1") +
 techSetParam("spacing2"))
 "metal2" "poly1")
```
Turning Parameter Evaluation Off

When you set a rule using a control parameter, the parameter is stored in the technology file. However, the `techGet` SKILL functions return the evaluated expression. To retrieve the control parameter itself, you must turn parameter evaluation off with `techSetEvaluate`. The following is part of an interactive session in the CIW that turns parameter evaluation off and retrieves the parameter itself.

```plaintext
Check the value of the rule. The CIW returns this value.
```

```
techGetSpacingRule(tf "minSpacing" "poly1" "metal2")
```

```
0.9
```

```
Turn parameter evaluation off.
```

```
techSetEvaluate(nil)
```

```
Check the rule again.
```

```
techGetSpacingRule(tf "minSpacing" "metal2" "poly1")
```

```
(techSetParam("spacing1") + techSetParam("spacing2"))
```

Remember to turn parameter evaluation back on. When it is off, none of your expressions or parameters are evaluated; they are read as strings.

The Layer Definitions Class

Adding a Layer

To add a layer to a technology file that is loaded in virtual memory with SKILL functions, you need to use several SKILL functions. This section defines which SKILL functions to use for what and presents a sample script that shows changes to the technology file.

To add a layer to the technology file, you use the SKILL functions listed in the following table:

<table>
<thead>
<tr>
<th>SKILL Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>techCreateLP</code></td>
<td>Creates a layer with default priority and display attributes</td>
</tr>
<tr>
<td><code>techSetLPArr</code></td>
<td>Updates the default values of all the display attributes except packet</td>
</tr>
<tr>
<td><code>techSetLPacketName</code></td>
<td>Updates the default packet assigned to the layer</td>
</tr>
<tr>
<td><code>techSetStreamLayer</code></td>
<td>If necessary, modifies the default Stream data created by <code>techCreateLP</code></td>
</tr>
</tbody>
</table>

April 2001 190 Product Version 4.4.6
The following is a portion of a script that creates a layer.

```
; Add a layer purpose pair
;===========================================
; Set the ID of the techfile in question
to tf

; Set the ID of the techfile in question
to tf

; Create layer
; (function returns t if creation is
; successful or layer already exists)
layer = techCreateLayer(tf 75 "echo" "eco")

; Create the purpose
; (function returns t if creation is
; successful or purpose already exists)
techCreatePurpose(tf 75 "myCell" "cel")

; Create lp using techfile, layer, and
; purpose from above
lp = techCreateLP(tf '("echo" "myCell") "echo")

; Update LP attributes as required.
; (LP creation sets them to defaults:
; priority == 0, all others == t)
techSetLPAttr(lp '(25 t t nil t nil))

; Update LP Packet name;
; (default packet name = "defaultPacket")
techSetLPPacketName(lp "bluesolid_L")

; Update Stream translation data;
; (default stream number = layer number,
; dataType = 0, translate = t)
techSetStreamLayer(tf '("echo" "myCell") 60 0 t)

Sets the ID of the technology file
to tf and, if not already loaded
in virtual memory, loads it.

Adds the following line in
techLayers:
( myCell 75 eco )

Adds the following line in
techPurposes:
( myCell 75 cel )

Appends the following line to the end
of the
techLayerPurposePriorities subclass and then moves it to
position 25 in the list:
( echo myCell )

Adds the following line to the
techDisplays subclass and
updates the defaults:
( echo myCell "bluesolid_L"
t t nil t )

Adds the following line in the
streamLayers subclass and
updates the Stream number
from 75 to 60.
```
The Layer Rules Class

Adding Via Layers

To add a via layer to the viaLayers subclass (section) of a technology file in virtual memory with SKILL functions, type the techSetViaLayer SKILL function in the CIW command line. If the viaLayers subclass does not exist, this function creates it. The following is part of an interactive session in the CIW that adds a via layer to the technology file in virtual memory.

```
Set the technology file database identifier to a simple variable.

tf = techGetTechFile(ddGetObj("echoTest"))

On the CIW command line, type techGetViaLayers to view existing via layers.

techGetViaLayers(tf)

The CIW returns this data.

Add a via layer.

techSetViaLayer(tf "metal2" "via2" "echo")

The CIW returns this value.

Verify that the new via layer has been added.

techGetViaLayers(tf)

The CIW returns this data.
```

Adding Equivalent Layers

To add a set of equivalent layers to the equivalentLayers subclass (section) of a technology file in memory with SKILL functions, type the techSetEquivLayer SKILL function in the CIW command line. If the equivalentLayers subclass does not exist, this function creates it.
The following is an example of the functions you can type into the CIW and the results you get:

Set the technology file database identifier to a simple variable.

Use `techGetEquivLayers` to view existing equivalent layers.

The CIW returns this data.

Add a set of equivalent layers.

The CIW returns this value.

Check the equivalent layers for the new set.

The CIW returns this data.

In this example, the `equivalentLayers` subclass does not exist and the `techGetEquivLayers` function returns `nil`. When you add a set of equivalent layers, the software automatically creates the subclass.

Set the technology file database identifier to a simple variable.

Use `techGetEquivLayers` to view existing equivalent layers.

The CIW returns this value.

Add a set of equivalent layers.

The CIW returns this value.

To check it, type this command.

The CIW returns this data.
The order of the layers is not important. If you try to add a duplicate set, the function returns True but does not add an additional set.

```plaintext
Use techGetEquivLayers to view existing equivalent layers. 

TechGetEquivLayers(tf)
```

The CIW returns this data.
```
("metal3" "echo")
```

Add the set in reverse order.
```
TechSetEquivLayer(tf list("echo" "metal3"))
```

The CIW returns this value.
```
True
```

The set of equivalent layers remains the same.
```
TechGetEquivLayers(tf)
```

The CIW returns this data.
```
("metal3" "echo")
```

### Adding Stream Layers

To add Stream translation data to the streamLayers subclass (section) of a technology file in memory with SKILL functions, type the `techSetStreamLayer` SKILL function in the CIW command line. If the streamLayers subclass does not exist, this function creates it.

**Note:** When you create a layer-purpose pair using `techCreateLP`, the system creates a Stream rule for the layer-purpose pair and sets the Stream layer number to the layer number.

The following is a portion of a script that uses `techSetStreamLayer` to update the Stream translation data of a layer.

```plaintext
; Update Stream translation data; 
; default stream number = layer number 
TechSetStreamLayer(tf `("echo" "myCell") 60 0 t)
```

### The Physical Rules Class

#### Adding Spacing Rules and Ordered Spacing Rules

To add rules to the spacingRules or orderedSpacingRules subclass (section) of a technology file in memory with SKILL, type the `techSetSpacingRule` or the `techSetOrderedSpacingRule` SKILL function in the CIW command line. If the subclass you want to update does not exist, these functions create it.
The controls class lets you create parameters that you can use to set rule values. See “Editing Class Data with SKILL Functions” on page 189 for an example of setting control parameters and using them to create a spacing rule with SKILL functions.

The Electrical Rules Class

Adding Characterization Rules and Ordered Characterization Rules

To add rules to the characterizationRules or orderedCharacterizationRules subclass (section) of a technology file in memory with SKILL, type the techSetElectricalRule SKILL function in the CIW command line. If the subclass you want to update does not exist, this function creates it.

The Virtuoso Layout Editor Rules Class

The following table lists the SKILL functions that operate on the Virtuoso Layout Editor Rules class of the technology file:

<table>
<thead>
<tr>
<th>SKILL Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>techSetLeLswLayers()</td>
<td>Overwrites the existing leLswLayers subclass with specified data</td>
</tr>
<tr>
<td>techSetLeLswLayer()</td>
<td>Appends the specified layer to the leLswLayers subclass</td>
</tr>
<tr>
<td>techGetLeLswLayers()</td>
<td>Returns the layers listed in the leLswLayers subclass</td>
</tr>
<tr>
<td>techIsLeLswLayer()</td>
<td>Returns t if the specified layer is listed in the leLswLayers subclass</td>
</tr>
</tbody>
</table>

The Virtuoso XL Rules Class

The following table lists the SKILL functions that operate on the Virtuoso XL Rules class of the technology file:

<table>
<thead>
<tr>
<th>SKILL Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>techSetLxExtractLayers()</td>
<td>Overwrites the existing lxExtractLayers subclass with specified data</td>
</tr>
</tbody>
</table>
The Virtuoso Compactor Rules Class

The following table lists the SKILL functions that operate on the Virtuoso Compactor Rules class of the technology file:

<table>
<thead>
<tr>
<th>SKILL Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>techSetCompactorLayers()</td>
<td>Overwrites the existing compactorLayers subclass with specified data</td>
</tr>
<tr>
<td>techSetCompactorLayer()</td>
<td>Appends the specified layer to the list in the compactorLayers subclass</td>
</tr>
<tr>
<td>techGetCompactorLayers()</td>
<td>Returns the layers listed in the compactorLayers subclass</td>
</tr>
<tr>
<td>techGetCompactorUsage()</td>
<td>Returns the Virtuoso® compactor keyword for the specified layer in the</td>
</tr>
<tr>
<td></td>
<td>compactorLayers subclass</td>
</tr>
<tr>
<td>techIsCompactorLayer()</td>
<td>Returns t if the specified layer is listed in the compactorLayers subclass</td>
</tr>
<tr>
<td>techSetSymWire()</td>
<td>Appends the specified wire to the symWires subclass</td>
</tr>
</tbody>
</table>
The Layout Synthesizer (LAS) Rules Class

The following table lists the SKILL functions that operate on the Layout Synthesizer Rules class of the technology file:

<table>
<thead>
<tr>
<th>SKILL Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>techGetSymWires()</code></td>
<td>Returns the wires and wire definitions listed in the <code>symWires</code> subclass</td>
</tr>
<tr>
<td><code>techGetSymWireParams()</code></td>
<td>Returns the definition of the specified wire listed in the <code>symWires</code> subclass</td>
</tr>
<tr>
<td><code>techSetSymRules()</code></td>
<td>Appends the specified symbolic rule to the <code>symRules</code> subclass</td>
</tr>
<tr>
<td><code>techGetSymRules()</code></td>
<td>Returns symbolic rules defined in the <code>symRules</code> subclass</td>
</tr>
<tr>
<td><code>techSetLasLayers()</code></td>
<td>Overwrites the existing <code>lasLayers</code> subclass with specified data</td>
</tr>
<tr>
<td><code>techSetLasLayer()</code></td>
<td>Appends the specified layer to the list in the <code>lasLayers</code> subclass</td>
</tr>
<tr>
<td><code>techGetLasLayers()</code></td>
<td>Returns the layers listed in the <code>lasLayers</code> subclass</td>
</tr>
<tr>
<td><code>techGetLasLayerByUsage()</code></td>
<td>Returns the layer assigned the specified keyword in the <code>lasLayers</code> subclass</td>
</tr>
<tr>
<td><code>techIsLasLayer()</code></td>
<td>Returns <code>t</code> if the specified layer is listed in the <code>lasLayers</code> subclass</td>
</tr>
<tr>
<td><code>techSetLasDevices()</code></td>
<td>Overwrites the existing <code>lasDevices</code> subclass with specified data</td>
</tr>
<tr>
<td><code>techSetLasDevice()</code></td>
<td>Appends the specified cellview and LAS keyword to the <code>lasDevices</code> subclass</td>
</tr>
<tr>
<td><code>techGetLasDevices()</code></td>
<td>Returns cellviews and keywords listed in the <code>lasDevices</code> subclass</td>
</tr>
<tr>
<td><code>techGetLasDeviceByName()</code></td>
<td>Returns the cellview assigned the specified keyword in the <code>lasDevices</code> subclass</td>
</tr>
</tbody>
</table>
The Place and Route Rules Class

The following table lists the SKILL functions that operate on the Place and Route Rules class of the technology file:

<table>
<thead>
<tr>
<th>SKILL Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>techIsLasDevice()</td>
<td>Returns t if the specified layer is listed in the lasDevices subclass</td>
</tr>
<tr>
<td>techSetLasWires()</td>
<td>Overwrites the existing lasWires subclass with specified data</td>
</tr>
<tr>
<td>techSetLasWire()</td>
<td>Appends the specified wire and LAS keyword to the lasWires subclass</td>
</tr>
<tr>
<td>techGetLasWires()</td>
<td>Returns the wire names and LAS keywords listed in the lasWires subclass</td>
</tr>
<tr>
<td>techGetLasWireByType()</td>
<td>Returns the wire assigned the specified keyword in the lasWires subclass</td>
</tr>
<tr>
<td>techSetLasProperties()</td>
<td>Overwrites the existing lasProperties subclass with specified data</td>
</tr>
<tr>
<td>techSetLasProperty()</td>
<td>Appends the specified LAS property and value to the lasProperties subclass</td>
</tr>
<tr>
<td>techGetLasProperties()</td>
<td>Returns the properties and values listed in the lasProperties subclass</td>
</tr>
<tr>
<td>techGetLasProperty()</td>
<td>Returns the value of the specified property listed in the lasProperties subclass</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SKILL Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>techSetPrRoutingLayers</td>
<td>Overwrites the existing prRoutingLayers subclass with the specified data</td>
</tr>
<tr>
<td>techSetPrRoutingLayer</td>
<td>Appends or updates the specified layer and direction in the prRoutingLayers subclass</td>
</tr>
<tr>
<td>techGetPrRoutingLayers</td>
<td>Returns the layers listed in the prRoutingLayers subclass</td>
</tr>
<tr>
<td>techGetPrRoutingDirection</td>
<td>Returns the routing direction for the specified layer</td>
</tr>
<tr>
<td>SKILL Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>techIsPrRoutingLayer</td>
<td>Returns true if the specified layer is listed in the prRoutingLayers subclass</td>
</tr>
<tr>
<td>techSetPrViaTypes</td>
<td>Overwrites the existing prViaTypes subclass with the specified data</td>
</tr>
<tr>
<td>techSetPrViaType</td>
<td>Appends or updates the specified via type in the prViaTypes subclass</td>
</tr>
<tr>
<td>techGetPrViaTypes</td>
<td>Returns the via and via type listed in the prViaTypes subclass</td>
</tr>
<tr>
<td>techGetPrViaType</td>
<td>Returns the via type of the specified device</td>
</tr>
<tr>
<td>techIsPrViaDevice</td>
<td>Returns true if the specified device is listed in the prViaType subclass</td>
</tr>
<tr>
<td>techSetPrStackVias</td>
<td>Overwrites the existing prStackVias subclass with the specified data</td>
</tr>
<tr>
<td>techSetPrStackVia</td>
<td>Appends or updates the specified layer in the prStackVias subclass</td>
</tr>
<tr>
<td>techGetPrStackVias</td>
<td>Returns the layers listed in the prStackVias subclass</td>
</tr>
<tr>
<td>techIsPrStackVia</td>
<td>Returns true if the specified device is listed in the prStackVias subclass</td>
</tr>
<tr>
<td>techSetPrMastersliceLayers</td>
<td>Overwrites the existing prMastersliceLayers subclass with the specified data</td>
</tr>
<tr>
<td>techSetPrMastersliceLayer</td>
<td>Appends or updates the specified layer to the prMastersliceLayers subclass</td>
</tr>
<tr>
<td>techGetPrMastersliceLayers</td>
<td>Returns the layers listed in the prMastersliceLayers subclass</td>
</tr>
<tr>
<td>techIsPrMastersliceLayer</td>
<td>Returns true if the specified device is listed in the prMastersliceLayers subclass</td>
</tr>
<tr>
<td>techSetPrViaRule</td>
<td>Updates the parameters for the specified via rule in the prViaRules subclass</td>
</tr>
<tr>
<td>techGetPrViaRules</td>
<td>Returns the via rules defined in the prViaRules subclass</td>
</tr>
<tr>
<td><strong>SKILL Function</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>techGetPrViaParams</td>
<td>Returns the parameters of the specified via rule defined in the <code>prViaRules</code> subclass</td>
</tr>
<tr>
<td>techSetPrGenViaRule</td>
<td>Updates the parameters for the specified generated via rule in the <code>prGenViaRules</code> subclass</td>
</tr>
<tr>
<td>techGetPrGenViaRules</td>
<td>Returns the generated via rules defined in the <code>prGenViaRules</code> subclass</td>
</tr>
<tr>
<td>techGetPrGenViaParams</td>
<td>Returns the parameters of the specified generated via rule defined in the <code>prGenViaRules</code> subclass</td>
</tr>
<tr>
<td>techSetPrTurnViaRule</td>
<td>Updates the parameters for the specified turn via rule in the <code>prTurnViaRules</code> subclass</td>
</tr>
<tr>
<td>techGetPrTurnViaRules</td>
<td>Returns the turn via rules defined in the <code>prTurnViaRules</code> subclass</td>
</tr>
<tr>
<td>techGetPrTurnViaParams</td>
<td>Returns the parameters of the specified turn via rule defined in the <code>prTurnViaRules</code> subclass</td>
</tr>
<tr>
<td>techSetPrNonDefaultRule</td>
<td>Updates the parameters for the specified nondefault rule in the <code>prNonDefaultRules</code> subclass</td>
</tr>
<tr>
<td>techGetPrNonDefaultRules</td>
<td>Returns the nondefault rules from the <code>prNonDefaultRules</code> subclass</td>
</tr>
<tr>
<td>techGetPrNonDefaultParams</td>
<td>Returns the parameters of the specified nondefault rule from the <code>prNonDefaultRules</code> subclass</td>
</tr>
<tr>
<td>techSetPrRoutingPitch</td>
<td>Appends or updates the specified layer in the <code>prRoutingPitch</code> subclass</td>
</tr>
<tr>
<td>techGetPrRoutingPitch</td>
<td>Returns the layers listed in the <code>prRoutingPitch</code> subclass</td>
</tr>
<tr>
<td>techSetPrRoutingOffset</td>
<td>Appends or updates the specified layer in the <code>prRoutingOffset</code> subclass</td>
</tr>
<tr>
<td>techGetPrRoutingOffset</td>
<td>Returns the layers listed in the <code>prRoutingOffset</code> subclass</td>
</tr>
<tr>
<td>techSetPrOverlapLayer</td>
<td>Appends the specified layer to the <code>prOverlapLayer</code> subclass</td>
</tr>
</tbody>
</table>
Checking a Technology File for Conformance to Cadence Application Requirements

After you have compiled your ASCII technology file, you can check your file for use with the Cadence® design framework II (DFII) applications. See “Checking a Technology File for Conformance to Application Requirements” on page 159 for details.

Discarding an Edited Technology File from Virtual Memory (Reloading Data from Disk)

If you have loaded an edited ASCII file and have decided not to keep your edits, you can reload the original technology file to virtual memory from disk.

1. From the CIW menu banner, choose Technology File – Discard.

   The Discard Edits To Technology File form appears.

   For a description of this form, see Appendix A.

   **Note**: If you are using a design management system and you have checked out the technology file, the Cancel Checkout button is also available.

2. In the Technology Library cyclic field, choose the library containing the technology file.
3. Click Cancel Checkout to discard your edits from both virtual memory and disk. The Cancel Checkout button is available only if you are using a design manager.

4. Click OK.

The Discard Edits dialog box appears, asking you to confirm the discard and reload the data saved on disk.

5. Click Yes.

The technology file on disk is loaded into virtual memory, deleting any changes you made since you last saved.

Note: With the Cadence Team Design Manager, if you chose Cancel Checkout, the version saved on disk is replaced with a link to the Team Design Manager repository and the version stored in the Team Design Manager system is loaded into virtual memory.

A message in the CIW indicates that the command was successful.

Technology file ‘cellTechLib’ was restored successfully.

Saving a Technology File Edited in Virtual Memory to Disk

To permanently save changes to a technology file edited in virtual memory, you must save the file to disk.

1. From the CIW menu banner, choose Technology File – Save.

   The Save Technology File form appears.

   ![Save Technology File Form](image)

   Choose the library containing the technology file.

   For a description of this form, see Appendix A.

2. From the Technology Library cyclic field, choose the library containing the technology file you want to save.

3. Click OK.

   A dialog box appears, asking you to confirm the save to disk. Click Yes.

   The technology file in virtual memory is saved to disk.
If you use a design management system, the following also occurs.

- If your environment is set to prompt you to check-in all or views, and you have checked out the technology file, the system prompts you to check-in the technology file.

- If your environment is set to check-in files or none, or you have not checked out the technology file, the system informs you that it cannot save the technology file to disk.

For more information about setting check-in options, see “Setting Automatic Checkout and Checkin Preferences” in the Design Framework II Help.

4. If you are prompted to check-in the technology file, choose a check-in option and click OK.

Choose yes or no.

If you choose yes, your changes are saved to disk and checked in.

If you choose no, your changes are saved to disk but are not checked in.

### About Unsaved Changes Message Boxes

Message boxes appear if you change the technology file (such as editing a layer-purpose pair) and then try to do any of the following before saving the change:

- Quit the software
- Change technology libraries
- Apply layer display changes
If you try to quit the software before saving your changes, you get a message box like the following example:

```
Save TechFiles

OK  Cancel  Help

Save these techfiles before closing?

cellTechLib  techfile.cds

All  None
```

➤ To save the changes, set the radio buttons and then click OK.
➤ To discard the changes, click Cancel.

If you try to change technology libraries before saving your changes, you get a message box like the following example:

```
Change Technology Library

Technology library cellTechLib has unsaved changes.
Save changes to cellTechLib?

Yes  No  Cancel  Help
```

➤ To save the changes and switch to the new technology library, click Yes.
➤ To discard the changes and switch to the new technology library, click No.
➤ To cancel the technology library change and revert back to the previously selected technology library, click Cancel.
If you make layer display changes and then click Apply, you get a message box like the following example:

![Apply Layer Purpose Pair Editor](image)

➤ To save the changes, click Yes.
➤ To discard the changes, click No.
Editing Class Data through CIW Pull-Down Menus: Controls and Layer Definitions

This chapter discusses the following:

- “Editing Controls Class Data” on page 207
- “Editing Layer Definitions Class Data” on page 210
Editing Controls Class Data

From the CIW pull-down menus, you can perform the following Controls class editing functions:

- Add a new techParams parameter
- Delete a techParams parameter
- Edit an existing techParams parameter

To add, delete, or edit Controls class technology file data in virtual memory, perform the following steps:

1. From the CIW menu banner, choose Technology File – Set Up.

   The Technology File Set Up form appears.

2. From the Technology Library cyclic field, choose the technology library containing the technology file to edit.

3. In the Classes list box, click on Control and, from the menu banner, choose File – Edit.

   or
In the *Classes* list box, double-click on *Control*.

The Technology File – Control form appears, displaying the Controls class data currently in the technology file.

For a description of this form, see Appendix A.
4. Edit the Controls class data as described below:

To add a techParams parameter, use either of the following methods:

Method 1
1. In the Name field at the bottom of the form, type the parameter name.
2. From the Value Type cyclic field, choose the value type.
3. In the Value field, type the value of the parameter.
4. Click Edit. The software adds the new parameter to the Existing Parameters list box.

Method 2
1. In the Existing Parameters list box, click on the parameter to use as a basis to edit to define a new parameter. The software displays the techParams arguments in the Name, Value Type, and Value data fields.
2. Edit the data in the data fields.
   Note: you must specify a new parameter name to produce a new parameter.
3. Click Edit.

To delete a techParams parameter:
1. In the Existing Parameters list box, click on the parameter to delete.
2. Click Delete. The software removes the parameter from the Existing Parameters list box.

To edit an existing techParams parameter:
1. In the Existing Parameters list box, click on the parameter to edit. The software displays the parameter definition arguments in the Name, Value Type, and Value fields at the bottom of the form.
2. Edit the parameter value in the Value field and choose a value type from the Value Type cyclic field.
3. Click Edit.

5. Click Apply or OK to save your changes to the virtual memory technology file.

To save your changes to the technology file on disk, choose Technology File – Save from the CIW menu banner.
Editing Layer Definitions Class Data

From the CIW pull-down menus, you can perform the following Layer Definitions class editing functions:

- **Add a new layer definition** *(layerDefinitions class)*
- **Edit the display packet assigned to a layer** *(layerDefinitions class, techDisplays subclass)*
- **Edit layer display attributes and Stream rules** *(layerDefinitions class, techDisplays subclass; layerRules class, streamLayers subclass)*
- **Rename layers or purposes** *(layerDefinitions class)*
- **Change layer priorities** *(layerDefinitions class, techLayerPurposePriorities subclass)*
- **Add or edit layer properties** *(layerDefinitions class, techLayerProperties subclass)*
- **Delete a layer** *(layerDefinitions class)*

Adding a Layer Definition

To add a layer definition to a technology file, do the following:

1. From the CIW menu banner, choose *Technology File – Edit Layers*.
   
The Layer Purpose Pair Editor form appears.
For a description of this form, see Appendix A.

Design manager users click here for more information.

2. From the Technology Library cyclic field, choose the library containing the technology file you want to edit.

The form is updated with layers from the technology file you chose.

3. Click the Add button.

The Add Layer Purpose Pair form appears.
For a description of this form, see Appendix A.

4. In the Layer Name field, type a layer name.

- When you type in an existing layer’s name, the Layer Purpose Pair Editor disables the Abbrev. field, replaces the Number cyclic field with a disabled text field, and displays the existing layer’s abbreviation and layer number.

- When you type in a new layer name, the Layer Purpose Pair Editor enables the Abbrev. field, displays a cyclic field listing the available layer numbers, and selects the first available number in that cyclic field. You can then type in a layer abbreviation of up to seven characters and/or change the layer’s number.

Note: The layer’s number uniquely identifies the layer and distinguishes it from other layers; the number does not affect the layer’s display priority.

Note: Applications that display layer names do not always have room to display the entire name. The optional abbreviation expands your control over what is displayed in narrow fields. Depending upon the width of the field for displaying the layer name, an application displays whichever of the following fits:
Technology File and Display Resource File User Guide
Editing Class Data through CIW Pull-Down Menus: Controls and Layer Definitions

- The full layer name
- The layer name truncated to fit (if no abbreviation is specified)
- The abbreviation
- The abbreviation truncated to fit

5. From the Purpose cyclic field, choose a purpose.

If the purpose you want is not listed in the cyclic field, it is not defined for the technology file. To add a new purpose, in the Add Layer Purpose Pair form, click Add Purpose. The Add Purpose form appears. Fill it in to specify the new purpose and click OK.

For a description of this form, see Appendix A.

Note: When specifying a purpose name and abbreviation, consider the fact that some applications display the layer and purpose names in selection windows, which often are of different widths or can be sized to different widths. The purpose name displayed in the window depends upon the width of the window and whether an abbreviation is specified, as follows:

- If there is room in the selection window, the application displays the full purpose name.
- If an abbreviation is specified and fits in the window, the application displays the abbreviation.
- If no abbreviation is specified or if an abbreviation is specified but is too long to fit in the window, the application displays the first and last characters of the full purpose name.

6. In the Add Layer Purpose Pair form, in the Display Resources list, click the display packet you want to assign.

To edit a display packet using the Display Resource Editor, click Edit Resources. For information about editing display packets, refer to “Changing a Display Packet Definition” on page 275.
7. Click the display attributes you want to set off and, if necessary, in the Translation Rules section, define the Stream translation data.

   For more information about display attributes and the translation rules, refer to the Add Layer Purpose Pair Form help.

   **Note:** If you want to add properties to the layer-purpose pair, you must first finish adding the layer and then add or edit the layer properties.

8. Click OK.

   The layer is added to the technology file and appears in the Layer Purpose Pair Editor and the Virtuoso® Layer Selection Window (LSW) or the Preview Object Selection Window (OSW).

9. To close the Layer Purpose Pair Editor, click Cancel.

   **Design manager users** click here for more information.

   **Note:** If you use the optional leLswLayers section in the Layout Editor Rules class of the technology file to determine how your layers appear in the LSW, you must manually add the new layer to that section before it will appear in the LSW. For more information about leLswLayers, refer to the Technology File and Display Resource File SKILL Reference Manual.

---

**Editing Layer Display**

There are two ways to edit how a layer appears on screen or in your plots. A layer is assigned a display packet that determines the fill and outline colors, stipple pattern, and line style used to display or plot the layer on a specific display device. To edit a layer’s display characteristics, you can do either of the following:

- Edit the display packet definition
- Assign a different display packet to the layer

For more information about editing display packets, refer to “Changing a Display Packet Definition” on page 275.

To assign a different display packet to the layer, do the following:

1. From the CIW menu banner, choose Technology File – Edit Layers.
   
   The Layer Purpose Pair Editor appears.
Editor Class Data through CIW Pull-Down Menus: Controls and Layer Definitions

For a description of this form, see Appendix A.

Design manager users click here for more information.

2. From the Technology Library cyclic field, choose the library containing the technology file you want to edit.

The form is updated with layers from the technology file you chose.

3. Click the layer you want to edit.

4. Click Edit.

The Edit Layer Purpose Pair form appears.
5. In the Display Resources list, click a different display packet to assign to the layer and click OK.

The layer is updated in virtual memory with the new display packet assignment. The layer icon is updated in the LSW and OSW.

6. To close the Layer Purpose Pair Editor, click Cancel.

Design manager users click here for more information.

**Editing Layer Attributes**

To edit the layer attributes, do the following:

1. From the CIW menu banner, choose Technology File – Edit Layers.

   The Layer Purpose Pair Editor appears. The following shows the part of the form you use:
2. From the Technology Library cyclic field, choose the library containing the technology file you want to edit.

   The form is updated with layers from the technology file you chose.

3. Click the layer you want to edit.

4. Click Edit.

   The Edit Layer Purpose Pair form appears. The following shows the part of the form you change:

<table>
<thead>
<tr>
<th>Priority</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selectable</td>
<td>■</td>
</tr>
<tr>
<td>Visible</td>
<td>■</td>
</tr>
<tr>
<td>Valid</td>
<td>■</td>
</tr>
<tr>
<td>Drag Enable</td>
<td>■</td>
</tr>
<tr>
<td>Change Layer</td>
<td>■</td>
</tr>
</tbody>
</table>

   Translation Rules
   - Translate streamLayer
   - Stream Data Type Number: 0
   - Stream Layer Number: 0

   Click the attributes and translation rules you want to change.

For a description of this form, see Appendix A.
5. Click the attributes you want to change.

6. If necessary, type new Stream translation values.

7. In the Edit Layer Purpose Pair form, click OK.
   The layer definition is updated in the technology file in virtual memory.

8. To close the Layer Purpose Pair Editor, click Cancel.
   Design manager users click here for more information.

Renaming Layers or Purposes

A layer is defined by a pair consisting of a layer name and a purpose name. A layer or purpose name can be shared among several layer-purpose pairs. If you change a layer or purpose name, the change affects all layers using it. You cannot rename system-defined layers and purposes. To rename a layer or purpose, do the following:

1. From the CIW menu banner, choose Technology File – Edit Layers.
   The Layer Purpose Pair Editor appears. The following shows the part of the Layer Purpose Pair Editor form you use:

   ![Diagram](Image)

   Choose the library containing the technology file you want to edit.
   Click the layer you want to edit.
   Click Edit.

   For a description of this form, see Appendix A.
   Design manager users click here for more information.

2. From the Technology Library cyclic field, choose the library containing the technology file you want to edit.
   The form is updated with layers from the technology file you chose.
3. Click the layer you want to edit and click _Edit_.

The Edit Layer Purpose Pair form appears with the existing layer and purpose data. The following shows the part of the form you change:

![Attributes](image)

Click _Rename_ to change a layer or purpose name.

For a description of this form, see _Appendix A_.

4. Click the _Rename_ button to the right of the layer or purpose name you want to change.

You cannot rename a system-defined layer or purpose.

The Rename Layer form or Rename Purpose form appears.

![Rename Layer](image)

Type a new name or abbreviation for the layer or purpose.

For a description of these forms, see _Appendix A_.

5. Type a new name or abbreviation for the layer or purpose.
Note: When specifying a layer or purpose name or abbreviation, consider the possibility of shortened names being displayed in selection windows.

6. Click OK.

   All layers using the layer name or purpose name are renamed in the technology file, the Layer Purpose Pair Editor, the Virtuoso Layer Selection Window (LSW), and the Preview Object Selection Window (OSW).

7. To close the Edit Layer Purpose Pair form, click OK.

8. To close the Layer Purpose Pair Editor form, click OK.

   Design manager users click here for more information.

Changing Layer Priorities

Swapping Priorities of Two Consecutive Layer-Purpose Pairs

You can switch the priority and relative position of two layer-purpose pairs listed consecutively in the Layer Purpose Pair Editor form as follows:

1. From the CIW menu banner, choose Technology File – Edit Layers.

   The Layer Purpose Pair Editor appears. The following shows the part of the Layer Purpose Pair Editor form you use:

   1. Choose the library containing the technology file you want to edit.

   2. With the left mouse button, click the first listed of the two layers you want to switch.

   3. With the middle mouse button, click the layer immediately following the first layer you selected.

   4. Click Move. The editor switches the positions of the two layers

For a description of this form, see Appendix A.

   Design manager users click here for more information.
2. From the Technology Library cyclic field, choose the library containing the technology file you want to edit.
   
   The form is updated with layers from the technology file you chose.

3. With the left mouse button, click the first (or highest priority) of the two layer-purpose pairs that you want to switch.
   
   To deselect, click the layer-purpose pair again with the middle mouse button

4. With the middle mouse button, click the next consecutive of the two layer-purpose pairs that you want to switch.
   
   To deselect, click the layer-purpose pair again with the middle mouse button.

5. Click Move.
   
   The layer-purpose pairs switch positions in the Layer Purpose Pair Editor. The priority numbers assigned to the pairs are updated in virtual memory to reflect the new order.

6. Click OK.
   
   Design manager users click here for more information.

Changing the Priority of a Layer-Purpose Pair

You can change the priority of a layer-purpose pair in the Layer Purpose Pair Editor form as follows:

1. From the CIW menu banner, choose Technology File – Edit Layers.
The Layer Purpose Pair Editor appears. The following shows the part of the Layer Purpose Pair Editor form you use:

1. Choose the library containing the technology file you want to edit.

2. With the left mouse button, click the layer you want to move.

3. With the middle mouse button, click the layer you want the first selected layer to follow.

4. Click Move.
   The editor moves the layer (pdiff nt) to the position selected (following pad dg).

For a description of this form, see Appendix A.

Design manager users click here for more information.

2. From the Technology Library cyclic field, choose the library containing the technology file you want to edit.
   The form is updated with layers from the technology file you chose.

3. With the left mouse button, click the layer-purpose pair whose priority you want to change.
   To deselect, click the layer-purpose pair again with the middle mouse button.

4. With the middle mouse button, click the layer-purpose pair you want the first layer-purpose pair to follow.
   To deselect, click the layer-purpose pair again with the middle mouse button.

5. Click Move.
   The editor moves the first layer-purpose pair to the position following the second layer-purpose pair you selected. The priority numbers assigned to the pairs are updated in virtual memory to reflect the new order.

6. Click OK.
   Design manager users click here for more information.
Adding or Editing Layer Properties

Cadence® design framework II (DFII) applications use rules in the technology file to determine how the applications manipulate objects on particular layers. Layer properties are user defined. To add or edit a layer property, do the following:

1. From the CIW menu banner, choose *Technology File – Edit Layers*.

   The Layer Purpose Pair Editor form appears. The following shows the part of the Layer Purpose Pair Editor form you use:

   1. Choose the library containing the technology file you want to edit.

   2. Click the layer you want to edit.

   3. Click *Edit*.

For a description of this form, see Appendix A.

2. From the *Technology Library* cyclic field, choose the library containing the technology file you want to edit.

   The form is updated with layers from the technology file you chose.

   *Design manager users* click here for more information.

3. Click the layer you want to edit.

4. Click *Edit*.

   The Edit Layer Purpose Pair form appears.
5. To add or edit properties for the layer, click *Set Properties*.

![Layer Property Editor form](image)

For a description of this form, see Appendix A.

The Layer Property Editor form appears. The Layer Property Editor form lets you add, modify, and delete properties set on a layer.

![Layer Property Editor form](image)

For a description of this form, see Appendix A.

6. To delete a property, choose the property name and click *Delete*.

When you click the property name, a blue box appears around it to show that it is selected.

When you click *Delete*, the property is removed from the form and from the technology file in virtual memory.

7. To add a property, click *Add*. 
The Add Property form appears.

![Add Property form](image)

These fields change, depending on the Type you choose.

For a description of this form, see Appendix A.

- Type a name, choose a type, and type a value.
- Click OK.

The property is added to the form and to the technology file in virtual memory.

8. To edit a property, click the property name and click *Modify*.

When you click the property name, a blue box surrounds the name and value to show they are selected.

When you click *Modify*, the Modify *property* form appears.

![Modify form](image)

For a description of this form, see Appendix A.

- Choose another type and type a new value.
Click OK.
The property is updated in the form and in virtual memory.

9. To close the Edit Layer Purpose Pair form, click OK.
10. To close the Layer Purpose Pair Editor, click OK.
    Design manager users click here for more information.

Editing Multiple Layer-Purpose Pairs

To select two or more layers for editing,

1. Click the first layer with the left mouse button
2. Click subsequent layers with the middle mouse button.
3. Click Edit.
   The Edit Layer Purpose Pair form appears containing information on the first layer-purpose pair you selected.
4. Edit data for the layer-purpose pair as desired.
5. Click Apply.
6. Click Next.
   The Edit Layer Purpose Pair form displays information on the next layer-purpose pair you selected.
7. Continue with steps 4 and 5 until you have edited the data for all of the layer-purpose pairs you selected.
   If you click Next after editing data for the last layer-purpose pair selected, the software displays the message:
   No more layer purpose pairs selected.
   To dismiss this message, click Close.
8. In the Editor Layer Purpose Pair form, click OK.
Deleting Layers

The layer is the most critical element in a technology file. All technology file classes can reference layers to create devices and specify how objects on different layers relate to each other. Do not delete a layer until you have removed all references to it from the technology file.

**Caution**

*If you delete a layer that is referenced in a device definition or rule, the system generates errors when it tries to place the device or apply the rule.*

To delete a layer or layers from the technology file, do the following:

1. From the CIW menu banner, choose Technology File – Edit Layers.

   The Layer Purpose Pair Editor form appears. The following shows the part of the Layer Purpose Pair Editor form you use:

   1. Choose the library containing the technology file you want to edit.

   2. Click the User button to view only user-defined layers.

   3. With the left mouse button, click the layer you want to delete. To delete multiple layers, click layers after the first one with the middle mouse button.

   4. Click Delete.

   For a description of this form, see Appendix A.

2. From the Technology Library cyclic field, choose the library containing the technology file you want to update.

   The form is updated with layers from the technology file you chose.

   *Design manager users* click here for more information.

3. To view only user-defined layers, click the User filter field.

   You cannot delete a system-defined layer.
4. Click the left mouse button on the layer you want to delete. To delete multiple layers, click the left mouse button on the first layer and the middle mouse button on subsequent layers.

5. Click *Delete*.

A dialog box appears, such as the following:

![Delete Layer Purpose Pair dialog box](image)

6. To delete the layer or layers, click *OK*.

The selected layers are deleted from the Layer Definitions class, but the layer name and purpose definitions still exist.
Editing Class Data through CIW Pull-Down Menus: Generic Rules

This chapter discusses the following:

- “Editing Layer Rules Class Data” on page 230
- “Editing Physical Rules Class Data” on page 236
- “Editing Electrical Rules Class Data” on page 240
Editing Layer Rules Class Data

From the CIW pull-down menus, you can perform the following Layer Rules class editing functions:

- Add a `viaLayers`, `equivalentLayers`, or `streamLayers` rule
- Delete a `viaLayers`, `equivalentLayers`, or `streamLayers` rule
- Edit the stream number, data type, or translate setting for an existing `streamLayers` rule

To edit Layer Rules class technology file data in virtual memory, perform the following steps:

1. From the CIW menu banner, choose *Technology File – Set Up*.
   
The Technology File Set Up form appears.

2. From the *Technology Library* cyclic field, choose the technology library containing the technology file to edit.

3. In the *Classes* list box, click on *Layer* and, from the menu banner, choose *File – Edit*.
   
   or
In the *Classes* list box, double-click on *Layer*.

The Technology File – Layer Rules form appears, displaying the Layer Rules class data currently in the technology file.

![Technology File – Layer Rules form](image)

For a description of this form, see Appendix A.

4. From the *Layer Rule* cyclic field, choose the subclass (section) to edit from the following:
   - Via Layers
   - Equivalent Layers
Stream Layers

The Technology File – Layer Rules form displays the subclass data currently in the technology file.

5. Edit the data as described below:

   viaLayers

To add a viaLayers rule, use either of the following methods:

*Method 1*

1. In the Layer 1, Via, and Layer 2 fields at the bottom of the form, type the layer names. (You can click Browse to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the appropriate Layer field.)
2. Click Edit. The software adds the new rule to the Existing Rules list box.

*Method 2*

1. In the Existing Rules list box, click on a rule to use as a basis to edit to define a new rule. The form displays the viaLayers arguments in the Layer 1, Via, and Layer 2 fields.
2. Edit the layer names in the Layer 1, Via, and Layer 2 fields. (You can use the Layer Browser as described in Method 1.)
3. Click Edit. The software adds the new rule to the Existing Rules list box.

To delete a viaLayers rule:

1. In the Existing Rules list box, click on the rule to delete.
2. Click Delete. The software removes the rule from the Existing Rules list box.
To add an equivalentLayers rule, use either of the following methods:

**Method 1**
1. In the Layer field, type the equivalent layers. (You can click Browse to view the names of the layers in the library with the Layer Browser. To choose multiple layers in the browser to specify as equivalent layers, hold down the Ctrl key while clicking on each of the equivalent layers.)
2. Click Edit. The software adds the new set of equivalent layers to the Existing Rules list box.

**Method 2**
1. In the Existing Rules list box, click on the rule to use as a basis to edit to define a new rule. The form displays the equivalentLayers arguments in the Layer field.
2. Edit the data in the Layer field. (You can use the Layer Browser as described in Method 1.)
3. Click Edit. The software adds the data to the Existing Rules list box.

To delete an equivalentLayers rule:
1. In the Existing Rules list box, click on the rule to delete.
2. Click Delete. The software removes the rule from the Existing Rules list box.
streamLayers

To add a streamLayers rule, use either of the following methods:

Method 1
1. In the fields at the bottom of the form, type the layer name or layer-purpose pair, stream number, and stream data type. (You can click Browse to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the Layer field.)
2. Choose the appropriate Translate selection.
3. Click Edit. The software adds the new set of equivalent layers to the Existing Rules list box.

Method 2
1. In the Existing Rules list box, click on a rule to use as a basis to edit to define a new rule. The form displays the streamLayers arguments in the Layer, Number, Data Type, and Translate fields.
2. Edit the data in the Layer, Number, Data Type, and Translate fields. Note: You must specify a new layer to produce a new streamLayers rule. (You can use the Layer Browser as described in Method 1.)
3. Choose the appropriate Translate selection.
4. Click Edit. The software adds the new rule to the Existing Rules list box.

To delete a streamLayers rule:
1. In the Existing Rules list box, click on the rule to delete.
2. Click Delete. The software removes the rule from the Existing Rules list box.

To edit an existing streamLayers rule:
1. In the Existing Rules list box, click on the rule to edit. The form displays the streamLayers arguments in the Layer, Number, Data Type, and Translate fields.
2. Edit the data in the Number and Data Type fields.
3. Choose the appropriate Translate selection.
4. Click Edit. The software changes the data for the layer in the Existing Rules list box.
6. Click *Apply* or *OK* to save your changes to the virtual memory technology file.

To save your changes to the technology file on disk, choose *Technology File – Save* from the CIW menu banner.
Editing Physical Rules Class Data

From the CIW pull-down menus, you can perform the following Physical Rules class editing functions:

- Add a spacingRules or orderedSpacingRules rule
- Delete a spacingRules or orderedSpacingRules rule
- Edit the value for an existing spacingRules or orderedSpacingRules rule
- Edit the value for the mfgGridResolution rule

To edit Physical Rules class technology file data in virtual memory, perform the following steps:

1. From the CIW menu banner, choose Technology File – Set Up.
   
   The Technology File Set Up form appears.

2. From the Technology Library cyclic field, choose the technology library containing the technology file to edit.
3. In the **Classes** list box, click on *Physical* and, from the form menu banner, choose *File – Edit*.  

   or

   In the **Classes** list box, double-click on *Physical*.

The Technology File – Physical Rules form appears, displaying the Physical Rules data currently in the technology file. For a description of this form, see Appendix A.
4. Edit the data as described below:

To add a spacing rule, use either of the following methods:

**Method 1**
1. Click on the appropriate radio button to choose 1 Layer to add a 1-layer spacing rule, 2 Layer to add a 2-layer spacing rule, or Ordered to add an ordered-spacing rule.
2. From the Rule cyclic field, choose the rule name.
3. In the fields at the bottom of the form, type the arguments for the new rule and choose the value type. (You can click Browse to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the appropriate Layer field.)
4. Click Edit. The software adds the new spacing rule to the Existing Rules list box.

**Method 2**
1. Click on the appropriate radio button to choose 1 Layer or 2 Layer to display spacingRules or to choose Ordered to display ordered-SpacingRules in the Existing Rules list box.
2. From the Rule cyclic field, choose the rule name. The software displays the rules of the selected type with the selected rule name in the Existing Rules list box.
3. In the Existing Rules list box, click on the rule to use as a basis to edit to define a new rule. The software displays the rule arguments in the data fields at the bottom of the form.
4. Edit the data in the data fields. **Note:** You must specify a new Rule or Layer name to produce a new rule. (You can use the Layer Browser as described in Method 1.)
5. Click Edit. The software adds the new rule to the Existing Rules list box.
To delete a spacing rule:
1. Click on the appropriate radio button to choose 1 Layer or 2 Layer for spacingRules or to choose Ordered for orderedSpacingRules.
2. From the Rule cyclic field, choose the rule name. The software displays the rules of the selected type with the selected rule name in the Existing Rules list box.
3. In the Existing Rules list box, click on the rule to delete.
4. Click Delete. The software removes the rule from the Existing Rules list box.

To edit an existing spacing rule:
1. Click on the appropriate radio button to choose 1 Layer or 2 Layer for spacingRules or to choose Ordered for orderedSpacingRules.
2. From the Rule cyclic field, choose the rule name. The software displays the rules of the selected type with the selected rule name in the Existing Rules list box.
3. In the Existing Rules list box, click on the rule to edit. The software displays the rule arguments in the data fields at the bottom of the form.
4. Edit the data in the Value Type or Value fields.
5. Click Edit. The software changes the data for the rule in the Existing Rules list box.

To edit mfgGridResolution:
1. From the Manufacturing Grid Resolution Value Type cyclic field, choose the value type.
2. In the Manufacturing Grid Resolution field, type the value.
3. Click Apply or OK to save your changes to the virtual memory technology file.

To save your changes to the technology file on disk, choose Technology File – Save from the CIW menu banner.
Editing Electrical Rules Class Data

From the CIW pull-down menus, you can perform the following Electrical Rules class editing functions:

- Add a \texttt{characterizationRules} or \texttt{orderedCharacterizationRules} rule
- Delete a \texttt{characterizationRules} or \texttt{orderedCharacterizationRules} rule
- Edit the value type or value for an existing \texttt{characterizationRules} or \texttt{orderedCharacterizationRules} rule

To edit Electrical Rules class technology file data in virtual memory, perform the following steps:

1. From the CIW menu banner, choose \textit{Technology File – Set Up}.
   
The Technology File Set Up form appears.

2. From the \textit{Technology Library} cyclic field, choose the technology library containing the technology file to edit.
3. In the *Classes* list box, click on *Electrical* and, from the menu banner, choose *File – Edit.*

   or

In the *Classes* list box, double-click on *Electrical.*

The Technology File – Electrical Rules form appears, displaying the Electrical Rules data currently in the technology file.

For a description of this form, see Appendix A.
4. Edit the data as described in the following:

To add an electrical rule, use either of the following methods:

**Method 1**
1. Click on the appropriate radio button to choose 1 Layer to add a 1-layer characterization rule, 2 Layer to add a 2-layer characterization rule, or Ordered to add an ordered characterization rule.
2. From the Rule cyclic field, choose the rule name.
3. In the fields at the bottom of the form, type the arguments for the new rule and select the value type. (You can click Browse to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the appropriate Layer field.)
4. Click Edit. The software adds the new characterization rule to the Existing Rules list box.

**Method 2**
1. Click on the appropriate radio button to choose 1 Layer or 2 Layer to display characterization-Rules or to choose Ordered to display orderedCharacterizationRules in the Existing Rules list box.
2. From the Rule cyclic field, choose the rule name. The software displays the rules of the selected type with the selected rule name in the Existing Rules list box.
3. In the Existing Rules list box, click on the rule to use as a basis to edit to produce a new rule. The software displays the rule arguments in the data fields at the bottom of the form.
4. Edit the data in the data fields. **Note:** You must specify a new Rule or Layer name to produce a new rule.
To delete an electrical rule:
1. Click on the appropriate radio button to choose 1 Layer or 2 Layer for characterizationRules or to choose Ordered for orderedCharacterizationRules.
2. From the Rule cyclic field, choose the rule name. The software displays the rules of the selected type with the selected rule name in the Existing Rules list box.
3. In the Existing Rules list box, click on the rule to delete.
4. Click Delete. The software removes the rule from the Existing Rules list box.

To edit an existing electrical rule:
1. Click on the appropriate radio button to choose 1 Layer or 2 Layer for characterizationRules or to choose Ordered for orderedCharacterizationRules.
2. From the Rule cyclic field, choose the rule name. The software displays the rules of the selected type with the selected rule name in the Existing Rules list box.
3. In the Existing Rules list box, click on the rule to edit. The software displays the rule arguments in the data fields at the bottom of the form.
4. Edit the data in the Value Type or Value fields.
5. Click Edit. The software changes the data for the rule in the Existing Rules list box.

5. Click Apply or OK to save your changes to the virtual memory technology file.

To save your changes to the technology file on disk, choose Technology File – Save from the CIW menu banner.
Editing Class Data through CIW Pull-Down Menus: Application-Specific Rules

This chapter discusses the following:

- “Editing Layout Editor Rules Class Data” on page 245
- “Editing Virtuoso XL Rules Class Data” on page 248
- “Editing Virtuoso Compactor Rules Class Data” on page 252
- “Editing Layout Synthesizer Rules Class Data” on page 258
Editing Layout Editor Rules Class Data

From the CIW pull-down menus, you can perform the following Layout Editor Rules class editing functions:

- Add a layer-purpose pair to the \texttt{leLswLayers} rule
- Delete a layer-purpose pair from the \texttt{lsLswLayers} rule

To edit Layer Rules class technology file data in virtual memory, perform the following steps:

1. From the CIW menu banner, choose \textit{Technology File} – \textit{Set Up}.

   The Technology File Set Up form appears.

2. From the \textit{Technology Library} cyclic field, choose the technology library containing the technology file to edit.

3. In the \textit{Classes} list box, click on \textit{Layout Editor} and, from the menu banner, choose \textit{File} – \textit{Edit}.

   or

   In the \textit{Classes} list box, double-click on \textit{Layout Editor}. 

   ![Technology File Set Up Form]

April 2001 245 Product Version 4.4.6
The Technology File – LE Rules (LswLayers) form appears, displaying the Layout Editor Rules data for the \texttt{leLswLayers} subclass currently in the technology file.

For a description of this form, see Appendix A.
4. Edit the data as described below:

To add a layer-purpose pair to the leLswLayers rule:
1. Select where in the list of layer-purpose pairs you want to add the new layer-purpose pair. In the Existing Rules list box, click on the layer-purpose pair just below where you want the new layer-purpose pair added. To add a layer-purpose pair to the bottom of the list, do not select a layer-purpose pair in the Existing Rules list box.
2. In the Layer field at the bottom of the form, type the layer name and purpose name of a layer-purpose pair not already displayed in the Existing Rules list box. (You can click Browse to view the names of the layer-purpose pairs in the library with the Layer Browser. When you click on a layer-purpose pair in the Layer Browser, the software automatically loads it into the Layer field.)
3. Click Insert. The software adds the new layer-purpose pair to the Existing Rules list box.

To delete a layer-purpose pair from the leLswLayers rule:
1. In the Existing Rules list box, click on the layer-purpose pair to delete.
2. Click Delete. The software removes the layer-purpose pair from the Existing Rules list box.

5. Click Apply or OK to save your changes to the virtual memory technology file.

To save your changes to the technology file on disk, from the CIW menu banner, choose Technology File – Save.
Editing Virtuoso XL Rules Class Data

From the CIW pull-down menus, you can perform the following Virtuoso® XL Rules class editing functions:

- Add a layer to the xlExtractLayers rule
- Add an xlNoOverlapLayers rule
- Delete a layer from the xlExtractLayers rule
- Delete an xlNoOverlapLayers rule

To edit Layer Rules class technology file data in virtual memory, perform the following steps:

1. From the CIW menu banner, choose Technology File – Set Up.
   
The Technology File Set Up form appears.

2. From the Technology Library cyclic field, choose the technology library containing the technology file to edit.

3. In the Classes list box, click on LX and, from the menu banner, choose File – Edit.
or

In the Classes list box, double-click on LX.

The Technology File – LX Rules form appears, displaying the Virtuoso XL Rules class data currently in the technology file.

For a description of this form, see Appendix A.

4. From the LX Rule cyclic field, choose the subclass (section) to edit from the following:

- Extract Layers
- No Overlap Layers

The Technology File – LX Rules form displays the subclass data currently in the technology file.
5. Edit the data as described below:

- **lxExtractLayers**

**To add an lxExtractLayers layer,** use either of the following methods:

**Method 1**
1. In the Layer field at the bottom of the form, type the layer name. (You can click Browse to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the Layer field.)
2. Click Edit. The software adds the new layer to the Existing Rules list box, which lists the layers to be monitored by the online extractor.

**Method 2**
1. In the Existing Rules list box, click on a layer name to use as a basis to edit to add a new layer. The software displays the layer name in the Layer field at the bottom of the form.
2. In the Layer field, edit the layer name. (You can use the Layer Browser as described in Method 1.)
3. Click Edit. The software adds the layer to the Existing Rules list box.

**To delete an lxExtractLayers layer:**
1. In the Existing Rules list box, click on the layer to delete.
2. Click Delete. The software removes the layer from the Existing Rules list box.
To add an lxNoOverlapLayers rule, use either of the following methods:

**Method 1**
1. In the Layer 1 and Layer 2 fields at the bottom of the form, type the layer names. (You can click Browse to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the appropriate Layer field.)
2. Click Edit. The software adds the new lxNoOverlapLayers rule to the Existing Rules list box.

**Method 2**
1. In the Existing Rules list box, click on a rule to use as a basis to edit to define a new rule. The software displays the rule arguments in the Layer 1 and Layer 2 fields at the bottom of the form.
2. In the Layer 1 and Layer 2 fields, edit the layer names. (You can use the Layer Browser as described in Method 1.)
3. Click Edit. The software adds the new lxNoOverlapLayers rule to the Existing Rules list box.

To delete an lxNoOverlapLayers rule:
1. In the Existing Rules list box, click on the rule to delete.
2. Click Delete. The software removes the rule from the Existing Rules list box.

6. Click Apply or OK to save your changes to the virtual memory technology file.

To save your changes to the technology file on disk, choose Technology File – Save from the CIW menu banner.
Editing Virtuoso Compactor Rules Class Data

From the CIW pull-down menus, you can perform the following Virtuoso Compactor Rules class editing functions:

- Add a compactorLayers, symWires, or symRules rule
- Delete a compactorLayers, symWires, or symRules rule
- Edit the arguments for an existing symWires rule

To edit Virtuoso Compactor Rules class technology file data in virtual memory, perform the following steps:

1. From the CIW menu banner, choose Technology File – Set Up.

   The Technology File Set Up form appears.

2. From the Technology Library cyclic field, choose the technology library containing the technology file to edit.

3. In the Classes list box, click on Compactor and, from the menu banner, choose File – Edit.
or

In the Classes list box, double-click on Compactor.

The Technology File – Compactor Rules form appears, displaying the Compactor Rules class data currently in the technology file.

For a description of this form, see Appendix A.

4. From the Compactor Rule cyclic field, choose the subclass (section) to edit from the following:
   - Layers
   - symWires
symRules

The Technology File – Compactor Rules form displays the subclass data currently in the technology file.

5. Edit the data as described below:

compactorLayers

To add a compactorLayers rule, use either of the following methods:

Method 1
1. In the Layer field at the bottom of the form, type the layer name. (You can click Browse to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the Layer field.)
2. From the Name cyclic field, choose the layer-usage keyword for the compactor layer.
3. Click Edit. The software adds the new compactorLayers rule to the Existing Rules list box.

Method 2
1. In the Existing Rules list box, click on a rule to use as a basis to edit to define a new rule. The form displays the compactorLayers arguments in the Name and Layer fields.
2. From the Name cyclic field, select the layer-usage keyword; in the Layer field, edit the layer name. (You can use the Layer Browser as described in Method 1.)
3. Click Edit. The software adds the new compactorLayers rule to the Existing Rules list box.

To delete a compactorLayers rule:
1. In the Existing Rules list box, click on the rule to delete.
2. Click Delete. The software removes the rule from the Existing Rules list box.
To add a **symWires** rule, use either of the following methods:

**Method 1**
1. In the bottom half of the form, turn the radio buttons on or off to choose whether or not to specify the optional **symWires** arguments for implant layer, width, legal region layer, and weighting and type the **symWires** arguments into the data fields. (You can click **Browse** to view the names of the layers in the library with the **Layer Browser**. When you click on a layer in the Layer Browser, the software automatically loads it into the appropriate **Layer** field.)
2. Click **Edit**. The software adds the new **symWires** rule to the **Existing Rules** list box.

**Method 2**
1. In the **Existing Rules** list box, click on the rule to use as a basis to edit to produce a new rule. The software displays the rule arguments in the data fields at the bottom of the form.
2. Edit the arguments in the data fields. (For layers, you can use the **Layer Browser** as described in **Method 1**.)
3. Click **Edit**. The software adds the new **symWires** rule to the **Existing Rules** list box.

**Note:** You must specify a new rule name to produce a new rule.
4. Click **Edit**. The software adds the new **symWires** rule to the **Existing Rules** list box.

To delete a **symWires** rule:
1. In the **Existing Rules** list box, click on the rule to delete.
2. Click **Delete**. The software removes the rule from the **Existing Rules** list box.
To edit an existing symWires rule:
1. In the Existing Rules list box, click on the rule to edit. The software displays the rule arguments in the data fields at the bottom of the form.
2. Edit the arguments in the data fields. Do not change the rule name. (You can click Browse to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the appropriate Layer field.)
3. Click Edit. The software changes the arguments for the rule in the Existing Rules list box.
To add a `symRules` rule, use either of the following methods:

**Method 1**
1. In the bottom half of the form, type the data into the data fields, turn the radio buttons on or off to choose whether or not to specify the optional `symRules` arguments for layer 2 and modifier, and select the rule type from the `Type` cyclic field. (You can click `Browse` to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the appropriate Layer field.)
2. Click `Edit`. The software adds the new `symRules` rule to the `Existing Rules` list box.

**Method 2**
1. In the `Existing Rules` list box, click on a rule to use as a basis to edit to produce a new rule. The software displays the rule arguments in the data fields at the bottom of the form.
2. Edit the arguments in the data fields. (For layers, you can use the Layer Browser as described in *Method 1*.)
3. Click `Edit`. The software adds the new `symRules` rule to the `Existing Rules` list box.

To delete a `symRules` rule:
1. In the `Existing Rules` list box, click on the rule to delete.
2. Click `Delete`. The software removes the rule from the `Existing Rules` list box.

6. Click `Apply` or `OK` to save your changes to the virtual memory technology file.

To save your changes to the technology file on disk, choose `Technology File – Save` from the CIW menu banner.
Editing Layout Synthesizer Rules Class Data

From the CIW pull-down menus, you can perform the following Layout Synthesizer Rules class editing functions:

- Add an `lasLayers`, `lasDevices`, `lasWires`, or `lasProperties` rule
- Delete an `lasLayers`, `lasDevices`, `lasWires`, or `lasProperties` rule
- Edit the layer name for an existing `lasLayers` rule
- Edit the cellview for an existing `lasDevices` rule
- Edit the wire name for an existing `lasWires` rule
- Edit the property value for an existing `lasProperties` rule

To edit Layout Synthesizer Rules class technology file data in virtual memory, perform the following steps:

1. From the CIW menu banner, choose *Technology File – Set Up*.

   The Technology File Set Up form appears.

![Technology File Set Up Form](image)
2. From the *Technology Library* cyclic field, choose the technology library containing the technology file to edit.

3. In the *Classes* list box, click on LAS and, from the menu banner, choose *File – Edit.*

   *or*

   In the *Classes* list box, double-click on LAS.

   The Technology File – LAS Rules form appears, displaying the LAS Rules class data currently in the technology file.

   ![Technology File – LAS Rules Form](image)

   For a description of this form, see Appendix A.
4. From the LAS Rule cyclic field, choose the subclass (section) to edit from the following:

- Layers
- Devices
- Wires
- Properties

The Technology File – LAS Rules form displays the subclass data currently in the technology file.

5. Edit the data as described below:
To add an lasLayers rule, use either of the following methods:

**Method 1**
1. Click on the radio buttons to choose whether to display Required, Optional, and Special Purpose layers in the Existing Rules list box and in the Name cyclic field.
2. From the Name cyclic field, choose a layer-usage keyword not already in the Existing Rules list box.
3. In the Layer field, type the layer name. (You can click Browse to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the Layer field.)
4. Click Edit. The software adds the new lasLayers rule to the Existing Rules list box.

**Method 2**
1. Click on the radio buttons to choose whether to display Required, Optional, and Special Purpose layers in the Existing Rules list box and the Name cyclic field.
2. In the Existing Rules list box, click on a rule to use as a basis to edit to define a new rule. The form displays the lasLayers arguments in the Name and Layer fields.
3. From the Name cyclic field, select a layer-usage keyword that is not already in the Existing Rules list box.
4. In the Layer field, edit the layer name if you want a different layer name. (You can use the Layer Browser as described in Method 1.)
5. Click Edit. The software adds the new lasLayers rule to the Existing Rules list box.

**To delete an lasLayers rule:**
1. Click on the radio buttons to choose whether to display Required, Optional, and Special Purpose layers in the Existing Rules list box and the Name cyclic field.
2. In the Existing Rules list box, click on the rule to delete.
3. Click Delete. The software removes the rule from the Existing Rules list box.
To edit an existing lasLayers rule:
1. Click on the radio buttons to choose whether to display Required, Optional, and Special Purpose layers in the Existing Rules list box and the Name cyclic field.
2. In the Existing Rules list box, click on the rule to edit. The form displays the lasLayers arguments in the Name and Layer fields.
3. In the Layer field, edit the layer name. (You can click Browse to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the Layer field.)
4. Click Edit. The software changes the layer in the rule in the Existing Rules list box.
To add an lasDevices rule, use either of the following methods:

Method 1
1. Click on the radio buttons to choose whether to display Required, Optional, and Special Purpose device types in the Existing Rules list box and the Name cyclic field.
2. From the Type cyclic field, select Transistors, Pins, or Contacts.
3. From the Name cyclic field at the bottom of the form, choose an LAS device keyword that is not already in the Existing Rules list box.
4. From the Cellview cyclic field, choose the cellview to which to apply the device keyword.
5. Click Edit. The software adds the new lasDevices rule to the Existing Rules list box.

Method 2
1. Click on the radio buttons to choose whether to display Required, Optional, and Special Purpose device types in the Existing Rules list box and the Name cyclic field.
2. From the Type cyclic field, select Transistors, Pins, or Contacts.
3. In the Existing Rules list box, click on a rule to use as a basis to edit to define a new rule. The form displays the lasDevices arguments in the Name and Cellview cyclic fields.
4. From the Name cyclic field, select a device keyword that is not already in the Existing Rules list box.
5. From the Cellview cyclic field, select the cellview.
6. Click Edit. The software adds the new lasDevices rule to the Existing Rules list box.

To delete an lasDevices rule:
1. Click on the radio buttons to choose whether to display required, optional, and special purpose device types in the Existing Rules list box and the Name cyclic field.
2. From the Type cyclic field, select Transistors, Pins, or Contacts.
3. In the Existing Rules list box, click on the rule to delete.
4. Click Delete. The software removes the rule from the Existing Rules list box.
To edit an existing lasDevices rule:
1. Click on the radio buttons to choose whether to display required, optional, and special purpose device types in the Existing Rules list box and the Name cyclic field.
2. From the Type cyclic field, choose the device type for the lasDevices rule to edit.
3. In the Existing Rules list box, click on the rule to edit. The form displays the data in the Name and Cellview cyclic fields.
4. From the Cellview cyclic field, choose a different cellview.
5. Click Edit. The software changes the rule in the Existing Rules list box.
lasWires

To add an lasWires rule, use either of the following methods:

Method 1
1. From the Name cyclic field at the bottom of the form, choose an LAS wire keyword that is not already in the Existing Rules list box.
2. From the Wires cyclic field, choose the wire name to which to apply the wire keyword.
3. Click Edit. The software adds the new lasWires rule to the Existing Rules list box.

Method 2
1. In the Existing Rules list box, click on a rule to use as a basis to edit to define a new rule. The software displays the lasWires arguments in the Name and Wires cyclic fields at the bottom of the form.
2. From the Name cyclic field, select a wire keyword that is not already in the Existing Rules list box.
3. From the Wires cyclic field, select the wire name.
4. Click Edit. The software adds the new lasWires rule to the Existing Rules list box.

To delete an lasWires rule:
1. In the Existing Rules list box, click on the rule to delete.
2. Click Delete. The software removes the rule from the Existing Rules list box.

To edit an existing lasWires rule:
1. In the Existing Rules list box, click on the rule to edit. The software displays the lasWires arguments in the Name and Wires fields at the bottom of the form.
2. From the Wires cyclic field, choose a different wire name.
3. Click Edit. The software changes the rule in the Existing Rules list box.
To add an \texttt{lasProperties} rule, use either of the following methods:

\textbf{Method 1}
1. From the \textit{Name} cyclic field at the bottom of the form, choose an LAS property keyword that is not already in the \textit{Existing Rules} list box.
2. In the \textit{Value} field, type the value for the property.
3. Click \textit{Edit}. The software adds the new rule to the \textit{Existing Rules} list box.

\textbf{Method 2}
1. In the \textit{Existing Rules} list box, click on a rule to use as a basis to edit to define a new rule. The form displays the \texttt{lasProperties} arguments in the \textit{Name} and \textit{Value} fields.
2. From the \textit{Name} cyclic field, select a property keyword that is not already in the \textit{Existing Rules} list box.
3. In the \textit{Value} cyclic field, type the value for the property if you want a different value.
4. Click \textit{Edit}. The software adds the new rule to the \textit{Existing Rules} list box.

To delete an \texttt{lasProperties} rule:
1. In the \textit{Existing Rules} list box, click on the rule to delete.
2. Click \textit{Delete}. The software removes the rule from the \textit{Existing Rules} list box.

To edit an existing \texttt{lasProperties} rule:
1. In the \textit{Existing Rules} list box, click on the rule to edit. The form displays the data in the \textit{Name} and \textit{Value} fields below.
2. In the \textit{Value} field, type a new value for the property.
3. Click \textit{Edit}. The software changes the rule in the \textit{Existing Rules} list box.

6. Click \textit{OK} or \textit{Apply} to save your changes to the virtual memory technology file.

To save your changes to the technology file on disk, choose \textit{Technology File – Save} from the CIW menu banner.
Editing, Reusing, and Merging Display Resources

This chapter discusses the following:

- “The Display Resource Updating Process” on page 268
- “Methods for Editing Display Resources” on page 269
- “Reusing a Display Resource File to Build a New Display Resource File” on page 269
- “Merging Display Resource Files” on page 270
- “Editing Display Resource Data through the CIW Pull-Down Menus” on page 274
- “Deleting Display Resources” on page 284
- “Editing Display Resource Data with SKILL Functions” on page 289
- “Reloading Source Display Resource Files” on page 291
- “Saving Display Resource Data to a File” on page 292
- “Testing a Display Resource File” on page 294
- “Editing a Saved Display Resource File” on page 295
- “About Save Changes Message Boxes” on page 297
The Display Resource Updating Process

The following summarizes the general process for updating or creating a new display resource file to add to your display resources:

- Create a new display resource file from scratch to merge with your existing display resources
- or
- Edit an existing display resource file with a text editor if you want to actually replace the data in that file with new data
- or
- Copy an existing display resource file to another file and edit it with a text editor to produce a new display resource file
- or

Select a location for the display resource file

File the display resource file in the selected location

Test the display resource file
Methods for Editing Display Resources

You can edit an existing display resource file, edit display resources during a design session, or create a new file from the data currently in virtual memory with the following methods:

■ Edit the ASCII file in a text editor and use the Display Resource Editor Load command to load it into virtual memory.

For more information about the syntax of the ASCII display resource file, refer to Chapter 6, “Creating a Display Resource File.”

■ Use the Display Resource Editor Load command to merge a new display resource file with the data already in virtual memory to use during the design session. You can also save all of the display resource data in virtual memory to a new display resource file on disk.

■ Use the Display Resource Editor to edit display resources in virtual memory. You can change the display packet definition; add colors, stipple patterns, and line styles; edit colors, stipple patterns, and lines styles; and add display devices in virtual memory to use during the design session. You can also selectively save either all of the display resource data in virtual memory or only the display resource data you have changed during a design session to a new display resource file on disk.

For more information about editing display resource data in virtual memory with the Display Resource Editor, see “Editing Display Resource Data through the CIW Pull-Down Menus” on page 274.

■ Use SKILL functions to load a display resource file and edit display resources in virtual memory. See “Editing Display Resource Data with SKILL Functions” on page 289 for a summary of the Cadence® SKILL functions available for manipulating display resources. For more information about the technology file SKILL functions, refer to the Technology File and Display Resource File SKILL Reference Manual.

Reusing a Display Resource File to Build a New Display Resource File

To reuse an existing display resource file to build a new one, you need only copy the existing file and edit it to your purposes. Be aware, however, of how it will be used when you choose a place to file it. See “How the Design Framework II Software Handles Multiple Display Resource Files” on page 29 for details.
Merging Display Resource Files

You can merge multiple source display resource files into one file or you can merge multiple display resource files in virtual memory when you are working on a design.

In either case, keep in mind that if a resource is defined in more than one of the display resource files you merge, the definition in the last file merged overrides any previous definitions. It is important, therefore, to be aware of the resources that are defined in the files and to merge the files in the order that results in the resource definitions you want.

The following example illustrates how merging replaces display resource definitions that are specified differently in more than one display resource file:

Files being merged, in order:

First file loaded for the merge.

Second file merged. When this file is merged, its definition of silver overwrites the definition of silver from the first file.

Third file merged. When this file is merged, its definition of magenta overwrites the definition of magenta from the second file.

Resultant merged file:
Merging Multiple Display Resource Files into One File

To merge multiple display resource files, perform the following steps:

1. From the CIW menu banner, choose Tools – Display Resources – Merge Files.
   The Merge Display Resource Files (DRF) form appears.

2. Identify the files you want to merge and the order in which you want them merged.
   Note: The order of selection is important. If a display resource is defined in more than one file being merged, the last definition merged overwrites any earlier ones.

3. Starting with the last file to merge and ending with the first file to merge, do one of the following for each file:
Click a library name in the From Library list box. The name and path of that library's display resource file appear in the Merge DRF files in sequence list box.

Enter a path and filename in the From File field and click Add. The path and filename appear in the Merge DRF files in sequence box.

Note: When you add a file, the software displays it at the bottom of the list in the Merge DRF files in sequence list box. The software merges the files listed in the list box from the bottom up. Consequently, the file you add first is merged last and the file you add last is merged first.

4. In the Destination DRF field, type the path and filename for the new file.

Note: The file must be named display.drf and must be placed in a location where the DFII initialization process can read it if you want it automatically loaded when you bring up the design software. Also take into account any other display.drf files loaded at initialization. See “How the Design Framework II Software Handles Multiple Display Resource Files” on page 29.

5. Make sure Load Merged DRF is on (the default) if you want the software to automatically load the file into virtual memory after merging.

If you do not load the display resource file at the same time as the merge, you must load it later from the Display Resource Editor to see the results.

6. Click OK or Apply.

The software merges the files into one, writes the resultant ASCII file to the location specified, and, if Load Merged DRF is on, loads the new file into virtual memory so that you can immediately see the results of the merge.

Merging New Display Resource Data into the Display Resource Data in Virtual Memory

When you load a display resource file, the data in the file is merged with the data already in virtual memory. To merge new display resource data into the display resource data in virtual memory, perform the following steps:

1. From the CIW menu banner, choose Tools – Display Resources – Editor.

   The Display Resource Editor form appears.

2. From the Display Resource Editor menu banner, choose File – Load.

   The Load form appears.
3. In the Files list box, click on a filename or type a path in the Selection field to select a file to load.

4. Click OK.

The software loads the file you specified into virtual memory and updates the Display Resource Editor with the new display resource data. If you have a cellview open, the software updates the layers only in the Layer Selection Window (LSW) or Object Selection Window (OSW).

To update the cellview display, do the following:

➤ From the menu on the cellview window, choose Window – Redraw.

If you want to use the display resource data now in virtual memory in the future, save it as a new display resource file. See “Saving Display Resource Data to a File” on page 292 for more information.

To return to the original set of display resource data, do the following:

➤ From the Display Resource Editor menu, choose File – Reinitialize.
Editing Display Resource Data through the CIW Pull-Down Menus

The Display Resource Editor, which is accessible through the CIW, allows you to edit the display resource data loaded in the current software session. Depending upon your system setup, this may be a conglomeration of multiple display resource files. See “How the Design Framework II Software Handles Multiple Display Resource Files” on page 29 for details.

To access the Display Resource Editor,

➤ Choose Tools – Display Resources – Editor from the CIW.

The Display Resource Editor appears.

For a description of this form, see Appendix A.
Display Resource Data Accessible through the CIW

You can edit most, but not all, display resource data loaded in the current software session with the Display Resource Editor (DRE). You can perform the following editing functions:

- Change the display packet definition
- Add colors, stipple patterns, and line styles
- Edit colors, stipple patterns, and line styles
- Add a display device

You cannot do the following with the Display Resource Editor (to perform these functions, you must edit the display.drf file with a text editor or, for the delete functions only, edit the display resource data in virtual memory with SKILL functions):

- Add a display packet
- Delete a display packet
- Delete a device, color, stipple, or line style

Once you have edited your display resource data, you can also do the following with the Display Resource Editor:

- Save your changes to disk in a display.drf file for use in later sessions
- Load additional display resource files into the current session
- Reload the original display resource data

Changing a Display Packet Definition

Display packets control how the design software displays and plots layers in your designs. A display packet can be shared among many layers. When you change the definition of a display packet, the design software automatically updates all layers that use the display packet.

Note: If you want your changes to be part of the software hierarchy so other users can access them,

➤ Edit the source display resource file (display.drf) that defines the display packet with a text editor.

To change a display packet definition loaded with the display resource data in the current software session, do the following:
1. From the CIW menu banner, choose Tools – Display Resources – Editor.

2. In the Display Resource Editor form, choose DRE from the Application cyclic field.
   
   The display packets defined for the software session appear in the left column of the form.

3. Click on the name of the display packet you want to edit.
   
   The Display Resource Editor highlights the display packet Fill Color, Outline Color, Stipple, and Line Style attributes.

4. Click the display resources you want for the display packet.

5. Click Apply.
   
   The software adds the data to the display resource file in virtual memory. It also updates the icon for the display packet in the Display Resource Editor with the new attributes. If you have a cellview open, the software updates the layers that use the display packet only in the LSW or OSW.

   To update the cellview display, do the following:
   
   ➤ From the menu in the cellview window, choose Window – Redraw.

   To make the changes permanent, use the Save form to save your changes to disk.

Changing a Layer Display

To change how a layer appears in your design, you edit the display packet assigned to the layer.

Note: Several layers can share the same display packet. When you change the display packet definition for one layer, you change the appearance of all other layers that use the same display packet.

Note: If you want your changes to be part of the software hierarchy so other users can access them,

   ➤ Edit the source display resource file (display.drf) that defines the display packet assigned to the layer with a text editor.

   To change the way a layer appears in the current software session, do the following:

   1. From the CIW menu banner, choose Tools – Display Resources – Editor.
      
      The Display Resource Editor form appears.
2. From the *Application* cyclic field, choose the application you use.

   The software updates the left column of the Display Resource Editor form to display the *Layers* list box with the layers defined in the technology file.

3. Click the layer you want to change.

   - The Display Resource Editor highlights the attributes for the display packet assigned to the layer.
   - The name of the display packet you are editing and all of the layers using it appear in the message bar at the bottom of the form. Click at the far right of the bar and press the right arrow on your keyboard to scroll the message bar.

4. Click the *Fill Style*, *Fill Color*, *Outline Color*, *Stipple*, and *Line Style* attributes you want for the layer.

   **Note:** The fill style you select overrides the stipple pattern and line style when there is a conflict.

5. Click *Apply*.

   The software adds the data to the display resource file in virtual memory. It also updates the icon for the layers that use the changed display packet in the Display Resource Editor form. If you have a cellview open, the software updates the layers only in the Layer Selection Window (LSW) or Object Selection Window (OSW).

   To update the cellview display, from the menu on the cellview window, choose *Window – Redraw*.

   To make these changes permanent, use the *Save form* to save your changes to disk.

**Adding Color**

Fill and outline colors come from the same set of colors. To add a new color to the set, do the following:

1. From the CIW menu banner, choose *Tools – Display Resources – Editor*.

   The Display Resource Editor form appears.

2. Under the *Fill Color* or *Outline Color* column, click *Edit*.

   The Fill Color Editor form appears, with a default color loaded in the *Swatch* area.
3. Click New.

4. In the Name field, type the name of the new color.

5. Do the following until the Swatch area shows the color you want:
   - From Example Colors, click a sample to start with.
   - Modify the color by moving the Red, Green, and Blue sliders, or type in red, green, and blue values in the text fields below the sliders.

6. Click OK.

The color appears in the form, and the software adds the data to the display resource data in virtual memory.

To make these changes permanent, use the Save form to save your changes to disk.
Editing Color

Fill and outline colors come from the same set of colors. To edit an existing color, do the following:

1. From the CIW menu banner, choose Tools – Display Resources – Editor.

   The Display Resource Editor form appears.

2. In the Fill Color or Outline Color column, double-click on a color.

   The Fill Color Editor form appears, with the color you selected loaded in the Swatch area.


3. Click Modify.

4. Move the Red, Green, and Blue sliders, or type new values in the text fields below the sliders.

5. Click OK.

   The software updates the color in the form and in the display resource data in virtual memory.

To update the cellview display, do the following:

   ➤ From the menu on the cellview window, choose Window – Redraw.

To make these changes permanent, use the Save form to save your changes to disk.

Adding a Stipple Pattern

To add a stipple pattern, do the following:

1. From the CIW menu banner, choose Tools – Display Resources – Editor.

   The Display Resource Editor form appears.

2. Under the Stipple column, click the icon of the stipple to edit.

   The Stipple Editor form appears, with the stipple you selected loaded in the Swatch area.

3. Under the Stipple column, click Edit.
The Stipple Editor appears.

Clear turns all pixels off. Invert reverses the selection of every pixel.

Click one to set the resolution of the grid for editing the stipple.

Click the pixels you want to turn on or off.

The swatch of the stipple pattern is updated as you click pixels.

For a description of this form, see Appendix A.

4. In the Stipple Editor form, click New.

5. In the Name field, type the name of the new stipple.

6. In the Size field, select a resolution to be applied in the Swatch area.

7. Click the pixels in the grid until the swatch shows the stipple pattern you want.
   - To clear all the pixels in the Swatch area, click Clear.
   - To reverse the selection of all pixels, click Invert.

8. Click OK.

The stipple appears in the form, and the software adds the data to the display resource data in virtual memory.

To make these changes permanent, use the Save form to save your changes to disk.
Editing a Stipple Pattern

To edit a stipple pattern, do the following:

1. From the CIW menu banner, choose Tools – Display Resources – Editor.
   
   The Display Resource Editor form appears.

2. Under the Stipple column, double-click the icon of the stipple to edit.
   
   The Stipple Editor form appears, with the stipple you selected loaded in the Swatch area.

3. Click the pixels in the grid until the Swatch area shows the stipple pattern you want.
   
   - To clear all the pixels in the swatch, click Clear.
   - To reverse the selection of all pixels, click Invert.
   - To start over, click the stipple name again in the Custom Stipples list box.
   - To erase your edits and select another stipple to edit, click a new name in the Custom Stipples list box.

4. Click OK.

   The software updates the stipple pattern in the form and in the display resource data in virtual memory.

To update the cellview display, do the following:

➤ From the menu on the cellview window, choose Window – Redraw.

To make these changes permanent, use the Save form to save your changes to disk.

Adding a Line Style

To add a line style, do the following:

1. From the CIW menu banner, choose Tools – Display Resources – Editor.

   The Display Resource Editor form appears.

2. Click on the icon of the line style to edit.

3. Under the Line Style column, click Edit.

   The Line Style Editor form appears, with the line style you selected loaded in the Swatch area.
4. Click New.
5. In the Name field, type the name of the new line style.
6. Select the editing grid resolution and line thickness.
7. Click the pixels in the grid until the Swatch area shows the line style you want.
8. Click OK.

The line style appears in the form, and the software adds the data to the display resource data in virtual memory.

To make these changes permanent, use the Save form to save your changes to disk.

**Editing a Line Style**

To edit a line style, do the following:

1. From the CIW menu banner, choose Tools – Display Resources – Editor.

   The Display Resource Editor form appears.

2. Double-click the icon of the line style to edit.

   The Line Style Editor form appears, with the line style you selected loaded in the Swatch area.
3. Click the pixels in the grid until the Swatch Window shows the line style you want.
   - To clear all the pixels in the swatch, click Clear.
   - To reverse the selection of all pixels, click Invert.
   - To start over, click the line style name again.
   - To erase your edits and select another line style to edit, click a new name in the Custom Line Styles list box.

4. Click OK.

The line style appears in the form and the software updates the data in the display resource data in virtual memory.

To update the cellview display, do the following:

➢ From the menu on the cellview window, choose Window – Redraw.

To make these changes permanent, use the Save form to save your changes to disk.

**Adding a Display Device**

A display device is a piece of hardware (for example, a monitor or plotter) that you use to display, plot, or print your designs. You can define a display packet to appear differently on different display devices. For a list of commonly used display devices, refer to “Commonly Used Display Devices” on page 146.

To add a display device, do the following:

1. From the CIW menu banner, choose Tools – Display Resources – Editor.

   The Display Resource Editor form appears.

2. Click New.

   The New Device form appears.

   ![New Device Form](image)

   For a description of this form, see Appendix A.
3. In the Device Name field, type the new device name.

4. Click OK.

   The display device name appears in the Display field of the Display Resource Editor form and the software loads it into the display resource data in virtual memory.

To make these changes permanent, use the Save form to save your changes to disk.

Deleting Display Resources

This section discusses deleting display resources, which you must always do with caution because they can be used by multiple technology libraries and with multiple designs.

Deleting Colors, Stipples, and Line Styles

Deleting a display resource such as a color, stipple, or line style can affect files requiring system administrator privileges. Consider the following before you begin:

- Display packets are defined by a set of display resources. If you delete a display resource, display packets using the resource become invalid, as do all of the layers that use those display packets.
- To preserve the affected display packets, you must update the display packet definitions to include an existing display resource.
- Because the system merges several source display resource files to create the display resource data you use, the color, stipple, or line style resource you want to delete might exist in more than one source display resource file. It might even be defined differently in the different source files.
- To completely delete the display resource, you must delete it from all of the source display resource files.
- Users of your software hierarchy can create display resource files in their home directories that can overwrite the company-provided data. These users will need to delete their personal display resource files and reinitialize their display resource data.

Finding Display Packets Affected by Deleting Display Resources

To determine which display packets are affected by deleting a display resource, do the following:

1. From the CIW menu banner, choose Tools – Display Resources – Editor.
The Display Resource Editor appears.

2. Choose File – Save.

   The Save form appears.

3. In the Selection field, type a temporary filename.

4. Click OK.
For example:

5. In a text editor, open the temporary file you have created and search for the name of the display resource you want to delete.

   Note: Line styles and stipples can have the same name.

   The following are samples of display resource definitions:

   - A color definition looks like this:
     
     ```plaintext
     ;( DisplayName ColorsName Red Green Blue )
     ( display yellow 255 255 0 )
     ```
A stipple definition looks like this:

```
;({ Displayname  StippleName  Bitmap
  { display  dots  (( 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 ))
    ( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
    ( 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 )
    ( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
    ( 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 )
    ( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
    ( 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 )
    ( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
    ( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
    ( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
  ) }
```

A line style definition looks like this:

```
;({ DisplayNameLineStyleSizePattern
  { displaywide1( 1 1 0 1 0 1 1 1 0 1 } )
```

6. Search for the display resource name again to find a display packet definition using the display resource.

A display packet definition looks like this:

```
;({ DisplayName PacketName Stipple LineStyle Fill Outline
  { display  grayslashwide  slash wide gray gray 
```

7. Write the name of the display packet in a list.

8. Repeat Step 6 and Step 7 until you return to the display resource definition.

9. Exit the text editor and use the list to edit the source display resource files.

Editing the Source Display Resource Files

Do the following for each source display resource file your company maintains. For information about where the source files exist, refer to “How the Design Framework II Software Handles Multiple Display Resource Files” on page 29.

**Note:** If you make a change to a local display file or a display file in your design management system, you need to inform the designers using your hierarchy. Designers can create display resource files in their home directories that can overwrite the company-provided data. These designers will need to delete their personal display resource files and reinitialize their systems.
1. Open the display resource file in a text editor.

2. Search for the name of the display resource to delete.

3. Delete the definition of the display resource.

   For example, to delete the wide line style, delete the definition from the beginning parenthesis through the ending parenthesis:

   ```plaintext
   ;( DisplayName LineStyle Size Pattern )
   ( display wide 1 (1 1 0 1 0 1 1 1 0 1) )
   ```

4. Using the list of display packet names you created in the procedure described in “Finding Display Packages Affected by Deleting Display Resources” on page 284, do the following:

   - Search for each display packet name.
   - Replace the name of the deleted display resource with an existing display resource.

   For example, if you are deleting the wide line style and replacing it with the solid line style, this

   ```plaintext
   ;( DisplayName PacketName Stipple LineStyle Fill Outline )
   ( display grayslashwide slash wide gray gray )
   ```

   becomes this

   ```plaintext
   ;( DisplayName PacketName Stipple LineStyle Fill Outline )
   ( display grayslashwide slash solid gray gray )
   ```

5. Save the file and exit the text editor.

6. Repeat this procedure for the other display resource files your company maintains.

7. Broadcast your changes to users of your hierarchy.

   The following is a sample broadcast message:

   ```plaintext
   Notice of technology change:
   The line style ”wide“ has been deleted from the design hierarchy. Display packets using this line style have been updated to use the “solid” line style. (Display packets determine how your layers appear on screen and in your plots.)
   If you use a display resource file in your home directory or work area, please update your file.
   If you need assistance call John Doe at 3340.
   ```
Note: You can also use SKILL functions to delete display resources in virtual memory. See “Editing Display Resource Data with SKILL Functions” on page 289 for more information.

Deleting a Display Packet

Technology files reference display packet names. Before you delete a display packet, make sure your technology files do not reference that display packet. As a precaution, consider leaving unused display packets in your display resource file for later use.

➤ If you are sure you want to delete a display packet, open the source display resource file with a text editor and delete the display packet definition.

To refresh your display resource data for the session, restart your software or, in the CIW, select the DRE File – Reinitialize command.

For information about where the source display resource files exist, refer to “How the Design Framework II Software Handles Multiple Display Resource Files” on page 29.

Note: You can also use SKILL functions to delete display resources in virtual memory. See “Editing Display Resource Data with SKILL Functions” on page 289.

Editing Display Resource Data with SKILL Functions

The following table lists the SKILL functions that return information about and operate on display resource data:

<table>
<thead>
<tr>
<th>SKILL Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drDeleteDisplay()</td>
<td>Deletes the specified display device from virtual memory.</td>
</tr>
<tr>
<td>drDeleteColor()</td>
<td>Deletes the definition of the specified color for the specified display device from virtual memory.</td>
</tr>
<tr>
<td>drDeleteLineStyle()</td>
<td>Deletes the specified line style from virtual memory.</td>
</tr>
<tr>
<td>drDeletePacket()</td>
<td>Deletes the definition of the specified display packet for the specified display device from virtual memory.</td>
</tr>
<tr>
<td><strong>SKILL Function</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>drDeleteStipple()</td>
<td>Deletes the definition of the specified stipple for the specified display device from virtual memory.</td>
</tr>
<tr>
<td>drFindPacket()</td>
<td>Reads virtual memory and returns a list of attributes of the specified display packet for the specified display device.</td>
</tr>
<tr>
<td>drGetColor()</td>
<td>Reads virtual memory and returns the display device; color name; and the red, green, blue, and blink values for the specified color.</td>
</tr>
<tr>
<td>drGetDisplay()</td>
<td>Reads virtual memory and returns the display device identifier of the specified display device name.</td>
</tr>
<tr>
<td>drGetDisplayIdList()</td>
<td>Reads virtual memory and returns a complete list of display device identifiers.</td>
</tr>
<tr>
<td>drGetDisplayName()</td>
<td>Reads virtual memory and returns the display device name of the specified display device identifier.</td>
</tr>
<tr>
<td>drGetDisplayNameList()</td>
<td>Reads virtual memory and returns a complete list of display device names.</td>
</tr>
<tr>
<td>drGetLineStyle()</td>
<td>Reads virtual memory and returns the display device name and the line style name, thickness, and pattern for the specified line style.</td>
</tr>
<tr>
<td>drGetLineStyleIndexByName()</td>
<td>Reads virtual memory and returns the line style index number for the specified line style for the specified display device.</td>
</tr>
<tr>
<td>drGetPacketList()</td>
<td>Reads virtual memory and returns a list of display packets that are defined for the specified display device.</td>
</tr>
<tr>
<td>drGetPacketAlias()</td>
<td>Reads virtual memory and returns a list of display packets that are aliased to the specified display packet.</td>
</tr>
<tr>
<td>drGetPacketFillStyle()</td>
<td>Reads virtual memory and returns the fill style number of the specified display packet for the specified display device.</td>
</tr>
</tbody>
</table>
Reloading Source Display Resource Files

If you change the display resource data in your current session and want to return to the data you started with, or if you have used a text editor to edit source display resource files that are merged together when you start the software, you can reload the display resource data from the source files.

To reload display resources from the display resource files the software automatically loads when it starts, do the following:

1. From the CIW menu banner, choose Tools – Display Resources – Editor.
   
   The Display Resource Editor appears.

   
   A dialog box appears, asking you to confirm the command.

3. Click OK.
   
   The software loads the display resource files on disk in sequence into virtual memory, just as it does when you first start the software.

To update the cellview display,

- From the cellview menu, choose Window – Redraw.

For more information about how display resource data is loaded, refer to “How the Design Framework II Software Handles Multiple Display Resource Files” on page 29.
Saving Display Resource Data to a File

You can save the display resource data loaded in the current software session to a text file that you can use in later sessions. You can save either all of the display resource data or only the changes you made during a design session. To use the new display resource file in later software sessions, you can place the file in your home directory or in the directory from which you start the software. By placing customized display resource data in your home directory, you have control over how your designs appear on screen and in plots.

**Note:** If you save all display resource data in virtual memory, the file you save after editing during a software session contains all display resource data loaded at the beginning of the session plus the changes made during the session. In this case, you may want to edit it to include only the data you need before you place it in your home directory. For example, if your company’s local `display.drf` defines a color called `redBlink`, the Cadence design framework II (DFII) software first loads that definition. If the `display.drf` in your home directory contains a customized definition of the `redBlink` color, the customized definition overwrites the company definition for the current session.

To save data loaded in the current session to a file, do the following:

1. From the CIW menu banner, choose **Tools – Display Resources – Editor**.
   The **Display Resource Editor form** appears.
2. Choose **File – Save**.
   The Save form appears.
3. To specify the file to which to save display resource data, click a file in the Files list box or type a path in the Selection field.

4. To save all of the display resource data currently loaded into virtual memory, leave Save Change off.

   To save only the display resource data changed during the current session, turn Save Change on.
5. Click OK.

The software creates the file you specified.

Now you can do any of the following:

- **Edit the saved file.**
- If you save all display resource data in virtual memory to the file, all display resource data loaded for the session is written to the file. Consequently, you might want to cut and paste the data you need to a smaller file.
- Place the file so it is automatically loaded every time you start the DFII software.
- Load the file manually into a DFII session.
- Discard your edits for the current session by reloading the original display resource data now that you have saved your changes.

## Testing a Display Resource File

To test a display resource data file, do the following:

1. From the CIW menu banner, choose **Tools – Display Resources – Editor**.

   The Display Resource Editor appears.

2. Choose **File – Reinitialize**.

   A dialog box appears, asking you to confirm the command.

3. Click **OK**.

   The software loads the display resource files on disk in sequence into virtual memory.

4. Select **File – Load**.

   The Load form appears.

5. In the **Selection** field, type the name of your edited file.

6. Click **OK**.

   If there are errors in your file, messages appear in the CIW.

7. Open a cellview and look for objects using your customized display resources.
Editing a Saved Display Resource File

To edit a saved display resource file to include only the data that you need to customize, do the following:

1. Make a copy of the saved file to keep as a backup.
2. Open the saved file in a text editor.
3. To keep a record of what the file contains, or why you have customized some display resources, insert comments in the file.

   For example:
   
   ; Created 1/24/96 for Emerald project.
   ; Changed red color to look tan.

4. Delete all lines except those defining the display resources you have customized.

   - To customize a display device, preserve the lines defining the display device and the colors, stipples, line styles, and display packets that use it. Here is an example of what to preserve for the display device itself:

     ```
     drDefineDisplay(
       ;( DisplayName   #Colors   #Stipple   #LineStyles)
       ( my_plotter 52 32 32 )
     )
     ```

   - To customize a color, preserve the lines shown in this example:

     ```
     drDefineColor(
       ;( DisplayName ColorName   Red   Green   Blue )
       ( display red 255 230 192 )
     )
     ```
To customize a stipple, preserve the lines shown in this example:

```plaintext
drDefineStipple(
  ;( DisplayName StippleName  Bitmap     )
  ( display  stripes  ( (1 1 1 1 1 1 1 1 1)
    (0 0 0 0 0 0 0 0 0 ) ) )
)
```

Stipple definitions can be many lines long.

To customize a line style, preserve the lines shown in this example:

```plaintext
drDefineLineStyle(
  ;( DisplayName   LineStyle   Size   Pattern   )
  ( display       fuzzy       4     (1 0 1 0 1 0 1 0 1 ) )
)
```

Definition of customized line style

To customize a display packet, preserve the lines shown in this example:

```plaintext
drDefinePacket(
  ;(Display PacketName Stipple LineStyle Fill Outline)
  (display blueHash_SB Hash solid blue blueBlink)
)
```

Definition of customized display packet

5. Save your changes and exit the text editor.
6. Test your customized resource data file.
7. When the file works as you expect, do one of the following:
❑ Paste the customized resource definitions into one of the source display resource files

❑ Place the file in your home directory or work area directory

Your changes are automatically loaded every time you start the DFII software.

About Save Changes Message Boxes

Both the Display Resource Editor and the SKILL functions change the data in virtual memory. If you have made changes and try to exit the Display Resource Editor or the application before saving the changes, one of the following dialog boxes appears.

The Saving Display Information dialog box appears if you try to exit the Display Resource Editor before saving your changes.

➤ To save your changes, click Yes.

The software displays the Save form.

➤ To discard your changes, click No.

➤ To cancel the Exit command, click Cancel.
The Save Display Information dialog box appears if you have made changes that you did not save when exiting the Display Resource Editor or if you have made changes with SKILL functions and try to exit the application before saving those changes.

➤ To discard your changes, click Cancel.
➤ To save your changes, type a filename in the text field and click OK.
This appendix presents descriptions of the user interface forms discussed in this user guide.
New Technology Library Form

**Technology Library Name** is the name of the new Cadence® design framework II (DFII) library to create.

**Load ASCII Technology File** is the name of the ASCII file you want to compile and install in the new DFII library.

**Load Existing Technology Library** is the name of the DFII library containing a technology file you want to copy and install in the new technology library.

**Directory (non-library directories)** is the directory in which you want to place the new DFII library.

**Design Manager** is the design management system you choose. The options displayed in this cyclic field are those that are available to you.

- **No DM** provides no design management features (such as checkin and checkout) for your library. *No DM* is the only option displayed if you have no design manager available to you.

- **<Design Manager Name>** lets you open your library under a design management system. (This option appears if you are running the tool in a managed work area under control of a design manager.) *Design Manager Name* is the name of the design manager available to you. For example, if you are running the tool in a Cadence Team Design Manager work area, *TDM* is displayed as an option in the cyclic field.

**SKILL Function to Display Form**

```skil
tcDisplayNewTechForm()
```

This SKILL function displays the New Technology Library form. It is equivalent to the CIW *Technology File – New* command.
Check Technology File Form

**Technology Library** is the name of the library containing the technology file you want to update. Select one of the libraries in your search path from this cyclic field.

**Application** lists the applications for which you can check the contents of the technology file. Click *Select All* to choose all of the applications listed or click on individual application you intend to use.

**Print Message Types** lets you select the types of messages you want to see.

- **Info** messages are statistical and progress messages.
- **Warn** messages indicate that the application cannot function efficiently.
- **Error** messages indicate that some functions of the application might not work.

**SKILL Function to Display Form**

tcDisplayCheckTechForm()

This SKILL function displays the Check Technology File form. It is equivalent to the CIW *Technology File – Check* command.
Attach Technology Library to Design Library Form

**Design Library** is the name of the DFII design library with which you want to use the technology file.

**Cell** lets you attach a technology file to a specified cell.

**View** lets you attach a technology file to a specified view of a cell. **Note:** You must fill in both a cell and a view name.

**Browse** displays the Library Browser and lets you alternatively select the design library, cell, and view name to use with the technology file.

**Attach To Technology Library** is the name of the library containing the technology file you want to use with the DFII library or design.

**SKILL Function to Display Form**

tcDisplayAttachTechForm()

This SKILL function displays the Attach Technology Library to Design Library form. It is equivalent to the CIW *Technology File – Attach To* command.
Dump Technology File Form

**Technology Library** is the name of the library containing the technology file you want to dump. Select one of the libraries in your search path from this cyclic field.

**Classes** lets you select specific classes to dump.

**Select All** selects all classes and indicates that you want to dump the entire technology file.

**ASCII Technology File** is the name of the ASCII file to create.

---

**Caution**

*Do not overwrite your golden ASCII technology file. Instead, dump your data to a temporary file. An ASCII file produced with the Dump command does not contain any of the comments or SKILL programs that your golden file might contain.*

---

If you click *OK* or *Apply* on either the Dump Technology File form or the By Class form, the system creates the ASCII file.

---

**SKILL Function to Display Form**

`tcDisplayDumpTechForm()`

This SKILL function displays the Dump Technology File form. It is equivalent to the CIW *Technology File – Dump* command.
Load Technology File Form

**ASCII Technology File** is the name of the text file you want to compile and load into virtual memory.

**Classes** lets you select specific classes to load.

**Select All** selects all classes and indicates that you want to load the entire file.

**Technology Library** is the name of the library containing the technology file you want to update. Select one of the libraries in your search path from this cyclic field.

**Merge** indicates that you want to combine the data in the ASCII technology file with the technology data already in virtual memory.

**Note:** If you select the Merge option, functions that are order-dependent are replaced. For functions that are not order-dependent, the data in your ASCII technology file is appended to the corresponding functions in the binary file.

**Replace** indicates that you want to overwrite technology data loaded into virtual memory with data from the ASCII technology file.

**SKILL Functions to Display Form**

```skill
tcDisplayLoadTechForm()
```

This SKILL function displays the Load Technology File form. It is equivalent to the CIW *Technology File – Load* command; it is also equivalent to the

```skill
tcDisplayCompTechForm()
```

This SKILL function displays the Load Technology File form. It is equivalent to the CIW *Technology File – Load* command; it is also equivalent to the

```skill
tcDisplayLoadTechForm()
```
Layer Purpose Pair Editor Form

**Technology Library** is the library that contains the technology file you want to edit. The layers defined in the technology file appear in the bottom of the Layer Purpose Pair Editor form.

**Save** saves your changes to the binary technology file on disk. (Until you click **Save**, changes made using this form are applied only to the file open in virtual memory.)

**Display Type** changes the icons next to the layer name to show how the layer appears in the display devices (monitors and plotters) you’ve defined.

**Add** displays the Add Layer Purpose Pair form, which lets you add a layer.

**Edit** displays the Edit Layer Purpose Pair form, which lets you edit the selected layer.

**Delete** deletes the selected layer.

**Move** lets you change the priority (shown by the position of the layer in the form) of layers. Priority affects

- The order in which layer purpose pairs are displayed in a cellview
- The order in which they appear in the Virtuoso Layer Selection Window (LSW) or the Preview Object Selection Window (OSW)

**Filter** lets you view user-defined layers, system-reserved layers, or all layers.

**Selectable All** and **None** change the selectability attribute for all layers.

**Note:** This change is written to virtual memory immediately and cannot be undone.

**Visible All** and **None** change the visibility attribute for all layers.

**Note:** This change is written to virtual memory immediately and cannot be undone.

The layers defined for the specified technology file are listed in the lower portion of the form. The icon shown with each layer represents the display packet defined for the specified display type.

- To select a layer for editing, click it.
- To select two or more layers for editing, click the first layer with the left mouse button and subsequent layers with the middle mouse button.
- To select two consecutive layers to switch their positions, click the first layer with the left mouse button and the next consecutive layer with the middle mouse button.
To select a layer to move and a position to move it to, click the layer to move with the left mouse button and the layer to move it after with the middle mouse button.

To deselect a selected layer, click it again with the middle mouse button.

To edit multiple layers in sequence, do the following:

- In the Layer Purpose Pair Editor, select the layers to edit and click Edit.
- In the Edit Layer Purpose Pair form, edit the layer shown and click Apply.
- In the Edit Layer Purpose Pair form, click Next and edit the next layer.
- Repeat.

s and v specify the selectability and visibility of each layer.
Technology File – Technology File Set Up Form

**File** pull-down menu lets you edit the selected class or close the setup form.

**Technology Library** is the library that contains the technology file currently selected for editing.

**Classes** list box displays the technology file classes you can edit via the user interface. Click once on a class to display its subclasses in the Rules list box. Click twice on a class to display the setup form for editing that class.

**Rules** list box displays the subclasses you can edit via the user interface for the currently selected class.

**SKILL Function to Display Form**

```skill
tcDisplaySetupTechForm()
```

This SKILL function displays the Technology File Set Up form. It is equivalent to the CIW *Technology File – Set Up* command.
Technology File – Layer Browser Forms

**Technology Library** is the library that contains the technology file currently selected for editing.

**Filters** lets you choose whether to display user-defined layers, system layers, or both.

**Layer, Layer 1, Layer 2, Via, Implant Layer, Legal Region Layer** displays the existing layers of the type in the heading and lets you choose a layer or layer-purpose pair by clicking on it in the list. For the Layer Rules `equivalentLayers` subclass, which specifies multiple layers, you can choose more than one layer by holding down the `Ctrl` key while clicking on each layer.
Discard Edits To Technology File Form

Technology Library is the name of the library containing the technology file you want to load from disk. From this cyclic field, select one of the libraries in your search path.

SKILL Function to Display Form

tcDisplayDiscardTechForm()

This SKILL function displays the Discard Edits To Technology File form. It is equivalent to the CIW Technology File – Discard command.
Save Technology File Form

Technology Library is the name of the library containing the technology file you want to save. From this cyclic field, select one of the libraries in your search path.

SKILL Function to Display Form

tcDisplaySaveTechForm()

This SKILL function displays the Save Technology File form. It is equivalent to the CIW Technology File – Save command.
Technology File – Control Form

**Technology Library** is the library that contains the technology file currently selected for editing.

**Existing Parameters** list box initially displays the data currently in the technology file for the `techParameters` subclass. As you edit, delete, and add data, this list box displays the changed data.

**Edit** applies any editing or addition you have made on the form. The software applies the changes only on the editing form; it does not write them to the technology file until you click **OK** or **Apply**.

**Delete** deletes the parameter currently selected in the **Existing Parameters** list box.

The bottom section of the form displays the data for the currently selected parameter and allows you to type data. The fields displayed are as follows:

- **Name** displays the name of parameter in the parameter definition currently selected and lets you type a parameter name.

- **Value Type** displays the value type for the parameter definition currently selected and lets you select a value type.

- **Value** displays the value assigned to the parameter in the parameter definition currently selected and lets you type a value.
Add Layer Purpose Pair Form

**Layer Name** specifies the name of the layer. Type the name of the layer you want to use. If you type a layer name that already exists, the abbreviation and number are updated when you click or type in another field.

**Abbrev.** specifies an abbreviation of the layer name. The abbreviation is optional and can be up to seven characters long.

**Note:** Applications that display layer names do not always have room to display the entire name. The optional abbreviation expands your control over what is displayed in narrow fields. Depending on the width of the field for displaying the layer name, an application displays whichever of the following fits:

- the full layer name
- the layer name truncated to fit (if no abbreviation is specified)
- the abbreviation
- the abbreviation truncated to fit

**Number** is the number assigned to the layer name.

- If you type a layer name that has not been used with this technology file before, select a number from the cyclic field.
- If you type a layer name that already exists, the layer number is updated when you click or type in another field.

**Purpose** specifies the name of the purpose you want to use with the layer name. Available defined purposes (those that have been defined but have not been combined with the layer) are listed in the cyclic field. For example, if you selected the layer *metal*, and a *metal drawing* layer purpose pair exists, the purpose *drawing* does not appear in the cyclic field.

To create a new purpose, click **Add Purpose**.

**Add Purpose** lets you add a new purpose to the technology file.

**Abbrev.** is an optional abbreviation for the purpose name. This field is for information only.

**Priority** is the priority in which the layer is displayed in a cellview and appears in the Virtuoso Layer Selection Window (LSW) or the Preview Object Selection Window (OSW). The default is 0, which causes the system to draw objects drawn with this layer purpose combination before all other objects.
Selectable indicates that you can select objects drawn on the layer.

Visible indicates that you can see objects drawn on the layer.

Valid indicates that the layer appears in the Virtuoso LSW or the Preview OSW.

Drag Enable indicates that you can see objects drawn on the layer as you move them.

Change Layer indicates to Diva that changes to objects on this layer must be tracked.

Translate Stream Layer indicates whether you want objects on this layer to be translated using the pipo or Stream translator.

Stream Data Type Number is the data type to which to translate the layer.

Stream Layer Number is the Stream layer to which to translate the layer.

Display Resources specifies the name of the display packet assigned to the layer. Select a packet from the list under the field.

Set Properties lets you add user-defined properties to the layer, using the Layer Property Editor form.

Edit Resources starts the Display Resource Editor, which lets you edit display packets.
Add Purpose Form

**Purpose Name** is the name of the new purpose.

**Abbreviation** specifies an abbreviation of the purpose name. The abbreviation is optional and can be up to seven characters long.

**Note:** Applications that display purpose names do not always have room to display the entire name. The optional abbreviation expands your control over what is displayed in narrow fields. Depending on the width of the field for displaying the purpose name, an application displays whichever of the following fits:

- the full purpose name
- the abbreviation
- the first and last letters of the full purpose name
Edit Layer Purpose Pair Form

**Layer Name** is the name of the layer you are editing. This field is for information only.

**Abbrev.** is an optional abbreviation for the layer name. This field is for information only.

**Number** is the number assigned to the layer name. This field is for information only.

**Rename** displays the Rename Layer Form to let you modify the layer name or abbreviation. Because a layer name can be combined with more than one purpose, changing the layer name affects all layer purpose pairs using the layer name.

**Purpose** is the name of the purpose used with the layer name. This field is for information only.

**Rename** displays the Rename Purpose Form to let you modify the purpose name or abbreviation. Because a purpose can be associated with more than one layer name, changing the purpose name affects all layer purpose pairs using the purpose.

**Abbrev.** is an optional abbreviation for the purpose name. This field is for information only.

**Priority** is the priority in which the layer is displayed in a cellview and appears in the Virtuoso Layer Selection Window (LSW) or the Preview Object Selection Window (OSW).

**Selectable** indicates that you can select objects drawn on the layer.

**Visible** indicates that you can see objects drawn on the layer.

**Valid** indicates that the layer appears in the Virtuoso LSW or the Preview OSW.

**Drag Enable** indicates that you can see objects drawn on the layer as you move them.

**Change Layer** indicates to Diva that changes to objects on this layer must be tracked.

**Translate streamLayer** indicates whether you want objects on this layer to be translated using the pip0 or Stream translator.

**Stream Data Type Number** is the data type to which to translate the layer.

**Stream Layer Number** is the Stream layer to which to translate the layer.

**Display Resources** specifies the name of the display packet assigned to the layer. Select a packet from the list under the field.

**Edit Resources** starts the Display Resource Editor, which lets you edit display packets.

**Set Properties** lets you add user-defined properties to the layer with the Layer Property Editor form.
Rename Layer Form and Rename Purpose Form

From shows existing information.

To shows information you can change.

Rename Layer Form

Name is the name of the layer.

Abbreviation specifies an abbreviation of the layer name. The abbreviation is optional and can be up to seven characters long.

Note: Applications that display layer names do not always have room to display the entire name. The optional abbreviation expands your control over what is displayed in narrow fields. Depending on the width of the field for displaying the layer name, an application displays whichever of the following fits:

- the full layer name
- the layer name truncated to fit (if no abbreviation is specified)
- the abbreviation
- the abbreviation truncated to fit

Number is the layer number. You cannot edit this field.

Rename Purpose Form

Name is the name of the purpose on the Edit Layer Purpose form. You cannot edit this field.

Abbreviation specifies an abbreviation of the purpose name. The abbreviation is optional and can be up to seven characters long.

Note: Applications that display purpose names do not always have room to display the entire name. The optional abbreviation expands your control over what is displayed in narrow fields. Depending on the width of the field for displaying the purpose name, an application displays whichever of the following fits:

- the full purpose name
- the abbreviation
- the first and last letters of the full purpose name
Layer Property Editor Form

**Name and Number** lists the name and number for the selected layer. You cannot edit this information here. If you want to modify this information, click **Cancel** and edit the information in the Edit Layer Purpose Pair form.

The properties you have added appear at the bottom of the form.

- To add a property, click **Add** at the top of the form.
- To delete a property, click the name of the property and then click **Delete** at the top of the form.
- To change the property name, you must delete the property and then add a new property with the correct name.
- To modify the default value, value type, values in the cyclic field, or the minimum or maximum value (shown in parentheses after the property name), click on the property name and then click **Modify** at the top of the form.
Add Property and Modify `<property_name>` Forms

**Name** is the name of the property. You can type in this field only when adding a property. Property names for a layer purpose pair must be unique.

**Type** is the data type of the value of the property. Your choices are

- `int` integer
- `float` floating-point number
- `string` any text (will be displayed in a cyclic field)
- `boolean` on or off
- `ILExpr` Not supported. Do not use. If you apply this property, it will not work and will be lost the next time you initiate a design session.
- `ILList` SKILL list
- `NLPExpr` Not supported. Do not use. If you apply this property, it will not work and will be lost the next time you initiate a design session.
- `fileName` any string, used as a file name
- `time` large integer indicating the number of seconds since Jan. 1, 1970, displayed as a date
- `hierProp` Not supported. Do not use. If you apply this property, it will not work and will be lost the next time you initiate a design session.

**Value** is the value of the property. For boolean properties, type `TRUE` or `FALSE`.

**Minimum** and **Maximum Value** is available for integer, floating-point, and time properties. It sets the minimum and maximum values. These values appear in the Layer Property Editor form in parentheses after the property name.

**Possible Choices** is available for string properties. It sets the possible values that appear in the cyclic field in the Layer Property Editor form. One of the values listed here must match the default shown in the **Value** field of the Add Property form.
Technology File – Layer Rules Form

**Technology Library** is the library that contains the technology file currently selected for editing.

**Layer Rule** lets you select the `layerRules` subclass you want to edit and displays the current data for that subclass in the *Existing Rules* list box.

**Existing Rules** list box initially displays the data currently in the technology file for the subclass selected with the *Layer Rule* cyclic field. As you edit, delete, and add data, this list box displays the changed data.

**Edit** applies any editing or addition you have made on the form. The software applies the changes only on the editing form; it does not write them to the technology file until you click *OK* or *Apply*.

**Delete** deletes the rule currently selected in the *Existing Rules* list box.

The bottom section of the form displays the data for the currently selected layer rule and allows you to type data. The fields displayed are as follows for each `layerRules` subclass:

- **Via Layers**
  - **Layer 1** displays the name of layer 1 in the rule currently selected and lets you type a layer 1 name.
  - **Via** displays the name of the via layer in the rule currently selected and lets you type a via layer name.
  - **Layer 2** displays the name of layer 2 in the rule currently selected and lets you type a layer 2 name.

- **Equivalent Layers**
  - **Layer** displays the names of the equivalent layers in the rule currently selected and lets you type equivalent layer names.

- **Stream Layers**
  - **Layer** displays the name of the layer in the stream layer rule currently selected and lets you type a layer name.
  - **Number** displays the stream number in the rule currently selected and lets you type a stream number.
  - **Data Type** displays the data type in the rule currently selected and lets you type a data type.
  - **Translate** lets you select whether or not to translate the layer.
Technology File – Physical Rules Form

Technology Library is the library that contains the technology file currently selected for editing.

Manufacturing Grid Resolution Value Type lets you select the value type for a mfgGridResolution subclass specification.

Manufacturing Grid Resolution lets you specify the value for a mfgGridResolution subclass specification.

1 Layer lets you select spacingRules that specify one layer.

2 Layer lets you select spacingRules that specify two layers.

Ordered lets you select orderedSpacingRules.

Rule lets you select the rule type.

Existing Rules list box initially displays the data currently in the technology file that corresponds to the 1 Layer, 2 Layer, Ordered, and Rule selections. As you edit, delete, and add data, this list box displays the changed data.

Edit applies any editing or addition you have made on the form. The software applies the changes only on the editing form; it does not write them to the technology file until you click OK or Apply.

Delete deletes the rule currently selected in the Existing Rules list box.

The bottom section of the form displays the data for the currently selected physical rule and allows you to type data. The following are the fields displayed:

Layer 1 displays the name of layer 1 in the rule currently selected and lets you type a layer 1 name.

Layer 2 displays the name of layer 2 in the rule currently selected and lets you type a layer 2 name. This field is not operable with spacingRules that specify one layer; it applies to spacingRules that specify two layers and to orderedSpacingRules.

Value Type displays the value type for the currently selected rule and lets you select the value type for the value for the rule.

Value displays the value for the currently selected rule and lets you specify a value.
Technology File – Electrical Rules Form

Technology Library is the library that contains the technology file currently selected for editing.

1 Layer lets you select characterizationRules that specify one layer.

2 Layer lets you select characterizationRules that specify two layers.

Ordered lets you select orderedCharacterizationRules.

Rule lets you select the rule type.

Existing Rules list box initially displays the data currently in the technology file that corresponds to the 1 Layer, 2 Layer, Ordered, and Rule selections. As you edit, delete, and add data, this list box displays the changed data.

Edit applies any editing or addition you have made on the form. The software applies the changes only on the editing form; it does not write them to the technology file until you click OK or Apply.

Delete deletes the rule currently selected in the Existing Rules list box.

The bottom section of the form displays the data for the currently selected electrical rule and allows you to type data. The following are the fields displayed:

Layer 1 displays the name of layer 1 in the rule currently selected and lets you type a layer 1 name.

Layer 2 displays the name of layer 2 in the rule currently selected and lets you type a layer 2 name. This field is not operable with characterizationRules that specify one layer; it applies to characterizationRules that specify two layers and to orderedCharacterizationRules.

Value Type displays the value type for the currently selected rule and lets you select the value type for the value for the rule.

Value displays the value for the currently selected rule and lets you specify a value.
Technology File – LE Rules (Lsw Layers) Form

Technology Library is the library that contains the technology file currently selected for editing.

Existing Rules list box initially displays the lswLayers data currently in the technology file. As you edit, delete, and add data, this list box displays the changed data.

Insert adds the layer-purpose pair displayed in the Layer field unless that layer-purpose pair is already in the Existing Rules list box. The software applies the changes only on the editing form; it does not write them to the technology file until you click OK or Apply.

Delete deletes the layer-purpose pair currently selected in the Existing Rules list box.

The bottom section of the form displays the currently selected layer-purpose pair and allows you to type data. The following are the fields displayed:

Layer displays the name of layer-purpose pair currently selected and lets you type a layer-purpose pair name.
Technology File – LX Rules Form

Technology Library is the library that contains the technology file currently selected for editing.

LX Rule lets you select the subclass, either lxExtractLayers or lxNoOverlapLayers, for editing Virtuoso XL rules.

Existing Rules list box initially displays the data currently in the technology file that corresponds to the LX Rule selection. As you edit, delete, and add data, this list box displays the changed data.

Edit applies any editing or addition you have made on the form. The software applies the changes only on the editing form; it does not write them to the technology file until you click OK or Apply.

Delete deletes the rule currently selected in the Existing Rules list box.

The bottom section of the form displays the data for the currently selected XL rule and allows you to type data. The fields displayed are as follows for each lxRules subclass:

- **lxExtractLayers**
  
  Layer displays the name of the layer in the rule currently selected and lets you type a layer name.

- **lxNoOverlapLayers**
  
  Layer 1 displays the name of layer 1 in the rule currently selected and lets you type a layer 1 name.

  Layer 2 displays the name of layer 2 in the rule currently selected and lets you type a layer 2 name.
Technology File and Display Resource File User Guide  
Form Descriptions

Technology File – Compactor Rules Form

Technology Library is the library that contains the technology file currently selected for editing.

Compactor Rule lets you select the subclass, either compactorLayers, symWires, or symRules, for editing Compactor rules.

Existing Rules list box initially displays the data currently in the technology file that corresponds to the Compactor Rule selection. As you edit, delete, and add data, this list box displays the changed data.

Edit applies any editing or addition you have made on the form. The software applies the changes only on the editing form; it does not write them to the technology file until you click OK or Apply.

Delete deletes the rule currently selected in the Existing Rules list box.

The bottom section of the form displays the data for the currently selected XL rule and allows you to type data. The fields displayed are as follows for each lxRules subclass:

- **compactorLayers**
  - **Name** displays the usage for the currently selected layer and lets you select the usage for a layer you are specifying or editing.
  - **Layer** displays the name of the layer in the rule currently selected and lets you type a layer name.
  - **Browse** displays the Layer Browser form and lets you select the layer to display in the Layer field.

- **symWires**
  - **Name** displays the wire name for the rule currently selected and lets you type a wire name.
  - **Layer** displays the name of layer in the rule currently selected and lets you type a layer name.
  - **Browse** displays the Layer Browser form and lets you select the layer to display in the Layer field.
  - **Implant** is set when the rule currently selected specifies an implant layer and constraints. This section lets you select whether or not to specify implant layer constraints and provides the following fields for doing so:
Layer displays the implant layer name in the rule currently selected and lets you type an implant layer name.

Enclosure displays the implant spacing in the rule currently selected and lets you type an implant spacing value for the distance by which the implant layer must enclose the wire layer.

Width is set when the rule currently selected contains wire width specifications. This section lets you select whether or not to specify wire width constraints and provides the following fields for doing so:

Default displays the default width in the rule currently selected and lets you type a default wire width.

Min displays the minimum allowable wire width in the rule currently selected and lets you type a minimum allowable wire width.

Max displays the maximum allowable wire width in the rule currently selected and lets you type a maximum allowable wire width.

Legal Region is set when the rule currently selected specifies a legal region. This section lets you select whether or not to specify a legal region and provides the following fields for doing so:

Layer displays the legal region layer name in the rule currently selected and lets you type a legal region layer name.

Inside is set when the legal region name in the rule currently selected is inside. This radio button lets you select the inside legal region name.

Outside is set when the legal region name in the rule currently selected is outside. This radio button lets you select the outside legal region name.

WLM is set when the rule currently selected specifies a weighting. This section lets you select whether or not a weighting applies to the rule and provides the following field for specifying it:

Weight displays the weighting for the rule currently selected and lets you type a weighting, or relative priority for minimizing the length of the wire.

symRules

Layer displays the name of the layer in the rule currently selected and lets you type a layer name.

Browse displays the Layer Browser form and lets you select the layer to display in the Layer field.
**Cell** displays the name of the master cell of the component in the rule currently selected and lets you type a cell name.

**View** displays the view name of the component in the rule currently selected and lets you type a view name.

**Layer 2** is set when the rule currently selected specifies data for a second layer. This section lets you select whether or not to specify data for a second layer and provides the following fields for doing so:

- **Layer 2** displays the name of the second layer in the rule currently selected and lets you type a layer name.
- **Cell** displays the name of the master cell of the component in the rule currently selected and lets you type a cell name.
- **View** displays the view name of the component in the rule currently selected and lets you type a view name.

**Type** displays the rule type (separation or enclosure) for the rule currently selected and lets you select either rule type.

**Value** displays the minimum separation of enclosure distance allowed for the rule currently selected and lets you type a distance value.

**Modifier** is set when a modifier places a restriction on the rule currently selected and lets you select whether or not to apply a modifier. Two cyclic fields with the following selections are provided for doing so:

- **None, sameNet, diffNet** displays the modifier setting for the rule currently selected and lets you select a modifier.
- **None, horizontal, vertical** displays the modifier setting for the rule currently selected and lets you select a modifier.
**Technology File – LAS Rules Form**

**Technology Library** is the library that contains the technology file currently selected for editing.

**LAS Rule** lets you select the subclass, either compactorLayers, symWires, or symRules, for editing Compactor rules.

**Existing Rules** list box initially displays the data currently in the technology file that corresponds to the LAS Rule selection. As you edit, delete, and add data, this list box displays the changed data.

**Edit** applies any editing or addition you have made on the form. The software applies the changes only on the editing form; it does not write them to the technology file until you click OK or Apply.

**Delete** deletes the rule currently selected in the Existing Rules list box.

The bottom section of the form displays the data for the currently selected XL rule and allows you to type data. On the Technology File – LAS Rules form, in some cases, there are differences in the top section of the form as well. The special fields and selections displayed are as follows for each lxRules subclass:

- **lasLayers**
  - **Required** lets you choose whether to display rules for required layers in the Existing Rules list box and in the Name cyclic field.
  - **Optional** lets you choose whether to display rules for optional layers in the Existing Rules list box and in the Name cyclic field.
  - **Special Purpose** lets you choose whether to display rules for special-purpose layers in the Existing Rules list box and in the Name cyclic field.
  - **Name** lets you select the LAS usage of the layer.
  - **Layer** displays the name of the layer in the rule currently selected and lets you type a layer name.

- **lasDevices**
  - **Required** lets you choose whether to display rules for required devices in the Existing Rules list box and in the Name cyclic field.
  - **Optional** lets you choose whether to display rules for optional devices in the Existing Rules list box and in the Name cyclic field.
**Form Descriptions**

**Special Purpose** lets you choose whether to display rules for special-purpose devices in the *Existing Rules* list box and in the *Name* cyclic field.

**Type** lets you select the device type.

**Name** lets you select the LAS name.

**Cellview** lets you select the cellview.

- **lasWires**
  
  **Name** lets you select the wire type.
  
  **Wires** lets you select the wire name.

- **lasProperties**
  
  **Name** lets you select the property name.
  
  **Value** displays the value of the property in the rule currently selected and lets you type a property value.
**Merge Display Resource Files (DRF) Form**

**Select DRF to merge** is the section in which you specify which display resource files to merge.

- **From Library** list box lets you select display resource files from your technology libraries.

- **From File** lets you type in the path and name of a display resource file to merge.

- **Add** adds the selected or typed-in display resource file to the bottom of the Merge DRF files in sequence list.

**Note:** You add display resource files to the list one at a time. Add the last file to merge first. (The order is important. If a display resource is defined in more than one file being merged, the last definition loaded overwrites any earlier ones.) Add the files so that the first file you add is the file to merge last and the last file you add is the file to merge first. (Files listed in Merge DRF files in sequence merge from the bottom up.)

**Merge DRF files in sequence** is the section which lists the files to be merged. After the fill in the form and choose **OK** or **Apply**, the software merges the files in this list, starting with the file at the bottom of the list and working up to the top of the list.

- **Delete** lets you delete a selected file from the list of files to be merged. To do so, click on the file in the list to select it, and then choose **Delete**.

**Destination DRF** lets you specify the path and name of the new display resource file to create from the merged files. The file name must be **display.drf**.

**Load Merged DRF** lets you load the display resource file immediately after merging, making it possible to see your changes as soon as you redraw the design.

**Note:** If you do not load the DRF when you do the merge, you must load it later from the Display Resource Editor to see the results.

**OK** or **Apply** initiates the merge.

**Cancel** closes the form without performing a merge.

**Defaults** clears the **From File** and **Destination DRF** fields and sets **Load Merged DRF** on.
Display Resource Editor (DRE) Form

**Application** lets you select the application for which you want to edit display resources. If you select *DRE*, the display packets defined in the display resource file appear in the left column of the form. If you select *Virtuoso*, the layer purpose pairs defined in the technology file appear.

**Device** lets you select from the devices defined in the display resource file. The layers and packets shown in the left column of the form are refreshed to show the packets defined for the selected device.

**New** lets you add new display devices.

**Note:** The other fields at the top of the form and in the left column vary, depending on the value you select for Application. If you select *Virtuoso* or *Preview*, the *Tech Lib Name*, *Layers*, *All*, and *LSW* or *OSW* fields appear. If you select *DRE*, the left column lists packets.

**Tech Lib Name** is the library containing the technology file that defines the layers you want to edit.

**Layers** if you selected the *Preview* from the Application cyclic field, lists the layers that appear in the OSW; if you selected *Virtuoso* from the Application cyclic field, lists the layers that appear in the LSW.

Click *All* to display the system-reserved layers as well. The layers are listed in priority order.

**All** lists all layers defined in the specified technology file, including system-reserved layers.

**LSW** lists only the layers that appear in the *Virtuoso Layer Selection Window*.

**OSW** lists only the layers that appear in the *Preview Object Selection Window*.

**Fill Style** lists the fill styles defined in the display resource file. Click a fill style to choose it. Note that the fill style overrides the stipple and line style.

**Fill Color** lists the colors defined in the display resource file. Double click a color to edit it.

**Outline Color** lists the colors defined in the display resource file. Double click a color to edit it.

**Stipple** lists the stipple patterns defined in the display resource file. Double click a stipple to edit it.

**Line Style** lists the line styles defined in the display resource file. Double click a line style to edit it.
Apply updates the display resource file with the values you selected for the specified layer or packet.

Edit displays an editor that lets you create or edit colors, stipples, or line styles.
Fill Color Editor, Outline Editor, and Color Editor Forms

**New** and **Modify** indicate whether you are creating a new color or editing an existing color.

**Custom Colors** are the colors already defined in the display resource file. Click on one to edit it.

**Name** is the name of the color to create or edit. When you are creating a new color, you can type in a name. When you are editing an existing color, you cannot edit the field.

**Red/Green/Blue** let you set the RGB values of the color. You can move the sliders or type in values to change the color in the *Swatch* area.

**Blink** indicates whether you want the color to blink.

**Swatch** displays the color defined by the Red, Green, and Blue fields. The hexadecimal value of the color is updated with the color.

**Example Colors** are the colors defined in your `.Xdefaults` file. Click on one to load it into the Swatch area and use it as a basis for a new or edited color.
Stipple Editor Form

New and Modify indicate whether you are creating a new stipple or editing an existing stipple.

Custom Stipples lists the stipples already defined in the display resource file. Click on one to edit it.

Name is the name of the stipple to create or edit. When you are creating a new stipple, you can type in a name. When you are editing an existing stipple, you cannot edit the field.

Swatch displays the stipple as it would appear in the display device. The Swatch area is updated as you click pixels in the grid.

Size indicates the resolution of the grid. The larger the numbers, the higher the resolution of your editing area.

Clear clears the grid so you can start over.

Invert reverses the display of pixels in the grid and Swatch area.
Line Style Editor Form

**New** and **Modify** indicate whether you are creating a new line style or editing an existing line style.

**Custom Line Styles** lists the line styles already defined in the display resource file. Click on one to edit it.

**Name** is the name of the line style to create or edit. When you are creating a new line style, you can type in a name. When you are editing an existing line style, you cannot edit the field.

**Size** indicates the resolution of the sample line. The larger the number, the more pixels you have in your editing area.

**Thickness** controls the thickness of the line.

**Swatch Window** displays the line style as it would appear in the display device. The *Swatch Window* is updated as you click pixels in the sample line.

**Clear** clears the sample line so you can start over.

**Invert** reverses the display of pixels in the sample line and *Swatch Window*. 
New Device Form

Device Name is the name of the display device to create.
Load Form or Save Form

**Filter** lets you specify the directories and files you want to see. You can use wildcards to view files with the same suffix or prefix. Type a path and click *Filter* at the bottom of the form to update the *Directories* and *Files* fields.

**Directories** lists the directories under the path specified in the *Filter* field.

**Files** lists the files that match the path specified in the *Filter* field.

**Save Change** appears only on the Save form. It lets you choose whether to save all of the display resource data in virtual memory or only the data you have changed during the session.

**Selection** is the path of the file you want to load or save.
System-Reserved Layers and Purposes

This appendix lists the

- System-reserved layers
- System-reserved purposes
## Layers

The following table shows system-reserved layers.

<table>
<thead>
<tr>
<th>Layer Name</th>
<th>Layer Purposes</th>
<th>Layer Number</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unrouted</td>
<td>drawing</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Unrouted</td>
<td>drawing1</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Unrouted</td>
<td>drawing2</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Unrouted</td>
<td>drawing3</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Unrouted</td>
<td>drawing4</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Unrouted</td>
<td>drawing5</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Unrouted</td>
<td>drawing6</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Unrouted</td>
<td>drawing7</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Unrouted</td>
<td>drawing8</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Unrouted</td>
<td>drawing9</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Row</td>
<td>drawing</td>
<td>201</td>
<td>Used for Preview Silicon Ensemble™ rows</td>
</tr>
<tr>
<td>Row</td>
<td>label</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>drawing, label</td>
<td>202</td>
<td>Used for floorplan/placement group; applied to all place-and-route tools</td>
</tr>
<tr>
<td>Cannotoccupy</td>
<td>boundary, drawing</td>
<td>203</td>
<td>Used for Preview Gate Ensemble®</td>
</tr>
<tr>
<td>Canplace</td>
<td>drawing</td>
<td>204</td>
<td>Used for Preview Gate Ensemble</td>
</tr>
<tr>
<td>hardFence</td>
<td>drawing</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>softFence</td>
<td>drawing</td>
<td>206</td>
<td></td>
</tr>
<tr>
<td>y0</td>
<td>drawing</td>
<td>207</td>
<td></td>
</tr>
<tr>
<td>y1</td>
<td>drawing</td>
<td>208</td>
<td></td>
</tr>
<tr>
<td>y2</td>
<td>drawing</td>
<td>209</td>
<td></td>
</tr>
</tbody>
</table>
Technology File and Display Resource File User Guide
System-Reserved Layers and Purposes

<table>
<thead>
<tr>
<th>Layer Name</th>
<th>Layer Purposes</th>
<th>Layer Number</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>y3</td>
<td>drawing</td>
<td>210</td>
<td></td>
</tr>
<tr>
<td>y4</td>
<td>drawing</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>y5</td>
<td>drawing</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>y6</td>
<td>drawing</td>
<td>213</td>
<td></td>
</tr>
<tr>
<td>y7</td>
<td>drawing</td>
<td>214</td>
<td></td>
</tr>
<tr>
<td>y8</td>
<td>drawing</td>
<td>215</td>
<td></td>
</tr>
<tr>
<td>y9</td>
<td>drawing</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>designFlow</td>
<td>drawing,</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td></td>
<td>drawing1-9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stretch</td>
<td>drawing</td>
<td>218</td>
<td></td>
</tr>
<tr>
<td>edgeLayer</td>
<td>drawing, pin</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>changedLayer</td>
<td>tool0, tool1</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>unset</td>
<td>drawing</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>unknown</td>
<td>drawing</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>spike</td>
<td>drawing</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>hiz</td>
<td>drawing</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>resist</td>
<td>drawing</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>drive</td>
<td>drawing</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>supply</td>
<td>drawing</td>
<td>227</td>
<td></td>
</tr>
<tr>
<td>wire</td>
<td>drawing, flight, label</td>
<td>228</td>
<td></td>
</tr>
<tr>
<td>pin</td>
<td>annotate,</td>
<td>229</td>
<td></td>
</tr>
<tr>
<td></td>
<td>drawing, label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>text</td>
<td>drawing,</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td></td>
<td>drawing1-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>device</td>
<td>annotate,</td>
<td>231</td>
<td></td>
</tr>
<tr>
<td></td>
<td>drawing,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>drawing1-2, label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>border</td>
<td>drawing</td>
<td>232</td>
<td></td>
</tr>
<tr>
<td>Layer Name</td>
<td>Layer Purposes</td>
<td>Layer Number</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------</td>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>snap</td>
<td>drawing</td>
<td>233</td>
<td></td>
</tr>
<tr>
<td>align</td>
<td>drawing</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>prBoundary</td>
<td>boundary, drawing, label</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>instance</td>
<td>drawing, label</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>annotate</td>
<td>drawing, drawing1–9</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>marker</td>
<td>error, warning</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>select</td>
<td>drawing</td>
<td>239</td>
<td></td>
</tr>
<tr>
<td>winActiveBanner</td>
<td>drawing</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>winAttentionText</td>
<td>drawing</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>winBackground</td>
<td>drawing</td>
<td>242</td>
<td></td>
</tr>
<tr>
<td>winBorder</td>
<td>drawing</td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>winBottomShadow</td>
<td>drawing</td>
<td>244</td>
<td></td>
</tr>
<tr>
<td>winButton</td>
<td>drawing</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>winError</td>
<td>drawing</td>
<td>246</td>
<td></td>
</tr>
<tr>
<td>winForeground</td>
<td>drawing</td>
<td>247</td>
<td></td>
</tr>
<tr>
<td>winInactiveBanner</td>
<td>drawing</td>
<td>248</td>
<td></td>
</tr>
<tr>
<td>winText</td>
<td>drawing</td>
<td>249</td>
<td></td>
</tr>
<tr>
<td>winTopShadow</td>
<td>drawing</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>grid</td>
<td>drawing, drawing1</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td>axis</td>
<td>drawing</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>hilite</td>
<td>drawing, drawing1–9</td>
<td>253</td>
<td></td>
</tr>
<tr>
<td>background</td>
<td>drawing</td>
<td>254</td>
<td></td>
</tr>
</tbody>
</table>
# Purposes

The following table lists system-reserved purposes.

<table>
<thead>
<tr>
<th>Layer Purpose</th>
<th>Purpose Number</th>
<th>Type of Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>255</td>
<td>Layer properties that you want all the other purposes of the layer to inherit (for example, assign the all purpose to the minSpacing property of the metal1 layer so that all the metal1 layer-purpose pairs inherit the same minimum spacing value)</td>
</tr>
<tr>
<td>annotate</td>
<td>240</td>
<td>Schematic text</td>
</tr>
<tr>
<td>boundary</td>
<td>250</td>
<td>Boundary layer purpose you assign to layers of a place-and-route design (place-and-route tools)</td>
</tr>
<tr>
<td>cell</td>
<td>254</td>
<td>Not used by the Cadence® design framework (DFII) tools</td>
</tr>
<tr>
<td>drawing</td>
<td>252</td>
<td>Default layer purpose for drawing layers</td>
</tr>
<tr>
<td>drawing1</td>
<td>241</td>
<td>Purpose used by the system-reserved layers</td>
</tr>
<tr>
<td>drawing2</td>
<td>242</td>
<td>Purpose used by the system-reserved layers</td>
</tr>
<tr>
<td>drawing3</td>
<td>243</td>
<td>Purpose used by the system-reserved layers</td>
</tr>
<tr>
<td>drawing4</td>
<td>244</td>
<td>Used by the Interactive Wire Editor</td>
</tr>
<tr>
<td>drawing5</td>
<td>245</td>
<td>Purpose used by the system-reserved layers</td>
</tr>
<tr>
<td>drawing6</td>
<td>246</td>
<td>Purpose used by the system-reserved layers</td>
</tr>
<tr>
<td>drawing7</td>
<td>247</td>
<td>Purpose used by the system-reserved layers</td>
</tr>
<tr>
<td>drawing8</td>
<td>248</td>
<td>Purpose used by the system-reserved layers</td>
</tr>
<tr>
<td>drawing9</td>
<td>249</td>
<td>Purpose used by the system-reserved layers</td>
</tr>
<tr>
<td>error</td>
<td>239</td>
<td>Error markers for rule checking</td>
</tr>
<tr>
<td>flight</td>
<td>238</td>
<td>Flightlines in a schematic cellview</td>
</tr>
<tr>
<td>label</td>
<td>237</td>
<td>Layers for text data</td>
</tr>
<tr>
<td>net</td>
<td>253</td>
<td>Translated interconnect layers, Diva® software extracted views</td>
</tr>
<tr>
<td>pin</td>
<td>251</td>
<td>Pin layers</td>
</tr>
</tbody>
</table>
## System-Reserved Layers and Purposes

<table>
<thead>
<tr>
<th>Layer Purpose</th>
<th>Purpose Number</th>
<th>Type of Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>tool0</td>
<td>236</td>
<td>Markers for Diva incremental DRC</td>
</tr>
<tr>
<td>tool1</td>
<td>235</td>
<td>Markers for Diva incremental DRC</td>
</tr>
<tr>
<td>warning</td>
<td>234</td>
<td>Warning markers for rule checking</td>
</tr>
</tbody>
</table>
Resolving Layer Errors

Layer errors can occur when you have a design containing cellviews from libraries that use different technology files.

If the layers in the two technology files are defined differently, when you open the design, it can contain a variety of errors (for example, your hierarchy could be wrong or the layer color, line style, stipple, or outline for a layer could be incorrect). You can have any combination of the following errors:

- The layer does not exist.
  Refer to “Adding a Layer to a Technology File” on page 344.

- The layers have conflicting layer numbers.
  Refer to “Resolving Inconsistent Layer Names” on page 349.

- The layer appearance is defined differently.
  Refer to “Fixing Layers That Do Not Appear Correctly” on page 351.

**Note:** This appendix covers how to correct layer errors. You can prevent these layer errors by attaching libraries to a shared technology file. A good example of the shared technology file is the one used in the *Cell Design Tutorial*. These libraries are in the following directory:

```
install_dir/samples/tutorials/le/cell_design
```
Adding a Layer to a Technology File

When a design and the cellviews it contains use different technology files, layer errors can occur. If a cellview uses layers that are not defined in the technology file attached to the design, you must add the layer definitions to your technology file.

There are two ways you can add a layer to a technology file: you can manually edit the technology file, or you can use the Display Resource Editor (DRE) and Technology File forms to change the technology file.

Adding a Layer by Manually Editing the Technology File

To check for missing layers and add a layer by manually editing the technology file, do the following:

1. To get an editable version of a technology file, do the following:
   a. From the CIW menu banner, choose Technology File – Dump.
      
      The Dump Technology File form appears.

      b. Set the Technology Library cyclic field to the technology library compiled from the file you want to edit. (This should be the top-level technology file.)

      c. Set layerDefinitions on.

      Note: If the technology file has leLSW rules and you are adding a valid layer, set layerDefinitions and leRules on.
All the other Classes buttons should be off.

d. In the ASCII Technology File field, type a filename.

e. Click OK.

A text editor window opens, containing the ASCII technology file.

2. Check the ASCII technology file for missing layer definitions.

3. Using the text editor window, add any missing layers in the techLayers subclass in your User-Defined Layers section.

   In the following example, the F+ layer is added to the techLayers User-Defined Layers section.

   ```
   techLayers(
   ;( LayerName Layer# Abbreviation )
   ;( --------- ------ ------------ )
   ;User-Defined Layers:
   ( ndiff 1 ndiff )
   ( pdiff 2 pdiff )
   ( F+ 4 F+ )
   ( pwell 6 pwell )
   )
   ```

4. To define the layer-purpose pair and set the layer priority, type the layer name in the techLayerPurposePriorities and the techDisplays subclasses.

   In the following example, the F+ layer is added to the techLayerPurposePriorities subclass.

   ```
   techLayerPurposePriorities(
   ;( LayerName Purpose )
   ;( --------- ------- )
   ( isolation drawing )
   ( F+ drawing )
   ( poly2 drawing )
   ( nwell net )
   )
   ```
In the following example, the F+ layer is added to the techDisplays subclass and its attributes are set as follows:

<table>
<thead>
<tr>
<th>Packet</th>
<th>gold</th>
<th>Part of the Change Layer</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visibility</td>
<td>yes</td>
<td>Drag Enable</td>
<td>yes</td>
</tr>
<tr>
<td>Selectability</td>
<td>yes</td>
<td>Valid</td>
<td>yes</td>
</tr>
</tbody>
</table>

Sets the F+ layer attributes

You can use only display packet names that are defined in your display.drf file. If you need to create a new display packet, refer to “Creating a New Display Packet” on page 351.

5. If you dumped the leLSW rules in Step 1, add the layer to the leLSWLayers subclass.
6. Save your technology file and exit the text editor window.
7. Load the edited technology file into memory by choosing, from the CIW menu banner, Technology File – Load.

The Load Technology File form appears.
8. In the Load Technology File form, do the following:

   a. In the ASCII Technology File field, type the name of the ASCII technology file you just edited.

   b. Set layerDefinitions on.

   c. From the Technology Library cyclic field, choose the technology library for this technology file.

   d. Set Merge on.

   e. Click OK.

9. Save these changes to disk by choosing, from the CIW menu banner, Technology File – Save.

The Save Technology File form appears.

   a. Set the Technology Library cyclic field to the technology library you want to save.

   b. Click OK.

**Adding a Layer Using Forms**

To check for missing layers and add a layer to the technology file using the user interface, do the following:

1. From the CIW menu banner, choose Technology File – Edit Layers.
The Layer Purpose Pair Editor form appears.

2. Set the *Technology Library* cyclic field to the technology library you want to edit.
3. Check the form for missing layers.
4. Click *Add*.

The Add Layer Purpose Pair form appears.
5. In the *Layer Name* field, type a layer name.

6. In the *Abbr.* field, type an abbreviation for the layer name.

7. From the *Number* cyclic field, choose a layer number.

8. From the *Purpose* cyclic field, choose a purpose.

9. In the *Priority* field, type the layer priority.

10. Set the *Selectable*, *Visible*, *Valid Drag Enable*, and *Change Layer* buttons appropriately.

11. Set the *Translate Stream Layer* button appropriately.

   If you choose translation, type the stream data type number and the stream layer number into those fields.

12. In the Display Resources list box, click on a display packet.

13. Click *OK*.

14. In the Layer Purpose Pair Editor form, click *OK*.

15. Save these changes to disk by choosing, from the CIW menu banner, *Technology File – Save*.

   The *Save Technology File form* appears.

16. Set the *Technology Library* cyclic field to the technology library you want to save.

17. Click *OK*.

### Resolving Inconsistent Layer Names

When a design and the cellviews it contains use different technology files, layer errors can occur. If multiple technology files assign the same layer number to different layer names, you must assign a different layer number to all but one of the layer names.

To identify and resolve inconsistent layer name errors, do the following:

1. To get an editable version of the technology files you want to check for conflicting layer name/number assignments, do the following for each:

   a. In the CIW, choose *Technology File – Dump*.

      The *Dump Technology File form* appears.
b. Set the Technology Library field to the technology library compiled from the technology file you want to change.

c. Set Select All on.

d. In the ASCII Technology File field, type a filename.

e. Click OK.

   A text editor window opens containing the ASCII technology file.

2. Check the technology files you are using for different layer names with the same layer number assigned.

3. Identify all layer numbers that are used in the technology files.

4. In one of the files, search for a layer name assigned to the same layer number as another layer in the other file(s).

5. In the techLayers subclass, change the Layer# field to one of the unused numbers.

   In the following example, the nwell layer number was changed.

   Changes the nwell layer number

   Changes the nwell layer number

   Changes the nwell layer number

6. Continue assigning new, unused layer numbers until there are no more conflicts.

7. Save each technology file and exit the text editor.

8. Do the following for each edited technology file to load it into memory:

   a. From the CIW menu banner, choose Technology File – Load.

      The Load Technology File form appears.

   b. In the ASCII Technology File field, type the name of the edited technology file.

   c. Set the Technology Library cyclic field to the technology library you want to reload.

   d. Set Select All on.

   e. Set Replace on.

   f. Click OK.
Technology File and Display Resource File User Guide
Resolving Layer Errors

Fixing Layers That Do Not Appear Correctly

When a design and the cellviews it contains use different technology files, layer errors can occur. If components (such as path, contacts) or the layers shown in the Layer Selection Window (LSW) do not have the correct color, line style, stipple, or outline, the two technology files are assigning different display packets to a layer. A display packet defines the color, line style, stipple, and outline attributes. To fix this error, you can do one of the following:

- Edit the display packet for the layer
- Select a different display packet for the layer
- Create a new display packet and assign it to the layer

Editing a Display Packet

You can use the Display Resource Editor to change a display packet. Refer to Chapter 12, “Editing, Reusing, and Merging Display Resources” for information about editing a display packet.

Note: Editing a display packet changes the appearance of all layers using that display packet.

Selecting a Different Display Packet

You can use the Layer Purpose Pair Editor to assign a different display packet to the layer. Refer to “Editing Layer Display” on page 214 for information about assigning a different display packet to a layer.

Creating a New Display Packet

To create a new display packet, do the following:

1. Using a text editor, open the display.drf file in your home directory.
2. Add the display packet name to the `drDefinePacket` section.
   The easiest way is to copy one of the lines and then change the display packet name and definition.

   In the following example, the **blue** display packet was added.

   Adds the **blue** display packet.

   ```
   drDefinePacket(
   ; ( DisplayName PacketName Stipple LineStyle Fill Outline )
   (display yellow blank solid yellow yellow )
   (display blue blank solid blue blue )
   (display tan blank solid tan tan )
   )
   ``

3. Save the `display.drf` file and exit the text editor.

4. From the CIW menu banner, choose *Tools – Display Resource – Editor.*
   The Display Resource Editor form appears.

5. Set the *Application cyclic* field to *DRE.*

   The Load Technology File form appears.

7. Choose the display.drf file and click OK.

   The new display packet appears in the Packets list.

8. In the Display Resource Editor form, select the new display packet and choose the correct fill, outline, stipple, and line style.

9. Save these changes by choosing File – Save.

   The Save Technology File form appears.

10. Choose the display.drf file and click OK.

   You can assign this new display packet to an existing layer (using the Layer Purpose Pair Editor form) or to a new layer (using the Add Layer Purpose Pair form).

Technology File and Display Resource File Examples

This appendix contains

- Technology File Example
- Display Resource File Example
Technology File Example

The following is an example of an ASCII technology file.

; Generated on Nov 14 07:50:20 1996

;******************************************************
; Controls DEFINITION
;******************************************************
controls(
  techParams(theta 2.0)
)
physicalRules(
  techDefineSpacingRule(
    (minWidth metal1 techGetParam("theta") * 2))
)

;******************************************************
; LAYER DEFINITION
;******************************************************
layerDefinitions(

techPurposes(
  ;( PurposeName Purpose# Abbreviation )
  ;( -------- ------- ******* )
  ;User-Defined Purposes:
  ;System-Reserved Purposes:
  ( drawing1 241 dr1 )
  ( pin 251 pin )
) ;techPurposes

techLayers(
  ;( LayerName Layer# Abbreviation )
  ;( -------- ------ ******* )
  ;User-Defined Layers:
  ( ndiff 1 ndiff )
  ( pwell 6 pwell )
  ;System-Reserved Layers:
  ( Unrouted 200 unRoute )
  ( Row 201 Row )
) ;techLayers

techLayerPurposePriorities(

April 2001 355 Product Version 4.4.6
; layers are ordered from lowest to highest priority
;left( LayerName Purpose 
;left( -------- ------- 
(left( nimplant drawing 
(left( nwell net 
(left( nwell drawing 
);

); techLayerPurposePriorities

techDisplays(
;left( LayerName Purpose Packet Vis Sel Con2ChgLy DrgEnbl Valid 
;left( -------- ------- ------ --- --- --------- ------- ----- 
(left( nimplant drawing cyan t t t t t t 
(left( nwell net yellow t t nil t nil 
(left( nwell drawing yellow t t t t t t 
);

); techDisplays

techLayerProperties(
;left( PropNameLayer1 [Layer2]PropValue 
(left( defaultWidth ndiff 1.000000 
(left( defaultWidth(pdiff drawing)1.000000 
);

); layerDefinitions

;******************************************************
; LAYER RULES
;******************************************************

layerRules(

streamLayers(
;left( layer streamNumber dataType translate 
;left( ----- ------------ ------- --------- 
(left( "ptap" "drawing" 0 0 nil 
(left( "ptap" "net" 0 0 nil 
(left( ndiff 0 0 nil 
);

); streamLayers

viaLayers(
;left( layer1 viaLayer layer2 
;left( ----- ----- ----- 
(left( poly1 cont metall 

TECH FILE AND DISPLAY RESOURCE FILE EXAMPLES

TECHNICAL FILE AND DISPLAY RESOURCE FILE USER GUIDE

APRIL 2001 357 PRODUCT VERSION 4.4.6

( metal1 via metal2 )
); viaLayers
equivalentLayers(
; ( list of layers )
; ( -------------- )
( vapox metal3 )
); equivalentLayers
)
); layerRules

;***********************
; DEVICES
;***********************
devices(
tcCreateCDSDeviceClass()
symEnhancementDevice(
; (name sdLayer sdPurpose gateLayer gatePurpose w l sdExt gateExt
; legalRegion)

(PTR pdiff drawing poly1 drawing 3 1 1.5 1
(outside pwell drawing))
);
;
; no syDepletion devices
;
symContactDevice(
; (name viaLayer viaPurpose layer1 purpose1 layer2 purpose2
; w l (row column xPitch yPitch xBias yBias) encByLayer1 encByLayer2 legalRegion)

(M1_P cont drawing metal1 drawing pdiff drawing
1 1 (1 1.5 1.5 center center) 0.5 0.5 _NA_)
)
tfcDefineDeviceProp(
; (viewName deviceName propName propValue)
(symbolic PTAP function "substrateContact")
(symbolic NTAP function "substrateContact")
)
symPinDevice(
    ; (name maskable layer1 purpose1 w1 layer2 purpose2 w2 legalRegion)
    (bigM1_pin t metal1 drawing 3 _NA_ _NA_ _NA_ _NA_)
    (pdiff_T nil pdiff drawing 1 _NA_ _NA_ _NA_ (outside pwell drawing))
    (ndiff_T nil ndiff drawing 1 _NA_ _NA_ _NA_ (inside pwell drawing))
  )

; ; no ruleContact devices
;

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++;
; Opus Symbolic Device Class Definition
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++;
; no other device classes

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++;
; Opus Symbolic Device Declaration
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++;
;
; ; no other devices
;
)

;devices

;**********************************************************************
; PHYSICAL RULES
;**********************************************************************
physicalRules(

  orderedSpacingRules(
    ;( rule layer1 layer2 value )
    ;( ---- ------ ------ ----- )
    ( minEnclosure "cellBoundary" "nwell" 0.1 )
    ( minEnclosure "ndiff" "cont" 0.5 )
  )

  spacingRules(
    ;( rule layer1 layer2 value )
    ;( ---- ------ ------ ----- )
    ( minSpacing "ndiff" 1.0 )
    ( minSpacing "pdiff" 1.0 )
  )
)
mfgGridResolution(
  ( 0.100000 )
) ;mfgGridResolution
) ;physicalRules

;*******************************************************************************
;  COMPACTOR RULES
;*******************************************************************************
comparatorRules(
  comparatorLayers(
    ( layer usage )
    ( ----- ----- )
    ( ndiff    "diffusion" )
    ( pdiff    "diffusion" )
  ) ;comparatorLayers
symWires(
  ;(name layer [(impLayer impSpacing)] [(default min max)] [(legalRegion regionLayer)] [WLM])
  ( metal2  ("metal2" "drawing") nil (0.6 nil nil) )
  ( metal1  ("metal1" "drawing") nil (0.6 nil nil) )
) ;symWires

) ;comparatorRules

;*******************************************************************************
;  LAS RULES
;*******************************************************************************
lasRules(
  lasLayers(
    ( layer usage )
    ( ----- ----- )
    ( ndiff    "ndiffLayer" )
    ( pdiff    "pdiffLayer" )
    ( ("pwell" "drawing") "pwellLayer" )
  ) ;lasLayers

  lasDevices(
    ;( cellview Las name )
    ;( ----- ------- )
  ) ;lasDevices
;lasRules

;*******************************
; LE RULES
;*******************************
leRules(

leLswLayers(
    ;( layer          purpose         )
    ;( -----          -------         )
    ( metall1Res      drawing         )
    ( text            drawing         )
) ;leLswLayers
)
;leRules

;*******************************
; P&R RULES
;*******************************
prRules(

prRoutingLayers(
    ;( layer           preferredDirection     )
    ;( -----           ------------------      )
    ( poly1            "halfRoute"            )
    ( metall1          "horizontal"           )
) ;prRoutingLayers

prMastersliceLayers(
    ;( layers : listed in order of lowest (closest to substrate) to
    ;       highest )
    ;( -------------------------------      )
    ( ndiff    pdiff                )
) ;prMastersliceLayers

) ;prRules
Display Resource File Example

The following is an example of a display resource (display.drf) file.

drDefineDisplay(
    ;( DisplayName )
    ( display

) ; -----------------------------------------------------------------
; ------ Display information for the display device ‘display’. ------
; -----------------------------------------------------------------
drDefineColor(
    ;( DisplayName    ColorName    Red   Green   Blue   Blink )
    ( display white    255    255    255 )
    ( display whiteB   255    255    255     t)
    ( display silver   217    230    255 )
    ( display cream    255    255    204 )
    ( display pink     255    191    242 )
    ( display magenta  255    0      255 )
    ( display lime     0      255    0    )
    ( display tan      255    230    191 )
    ( display cyan     0      255    255 )
    ( display gray     204    204    217 )
    ( display grayB    204    204    217     t)
    ( display yellow   255    255    0    )
    ( display yellowB  255    255    0     t)
    ( display orange   255    128    0    )
    ( display red      255    0      0    )
    ( display purple   153    0      230 )
    ( display green    0      204    102 )
    ( display brown    191    64     38  )
    ( display blue     0      0      255 )
    ( display slate    140    140    166 )
    ( display gold     217    204    0    )
    ( display maroon   230    31     13  )
    ( display violet   94     0      230 )
    ( display forest   38     140    107 )
    ( display chocolate128    38     38  )
    ( display navy     51     51     153 )
)
( display  black       0    0    0    )
( display  winBack    224  224  224    )
( display  winFore    128   0    0    )
( display  winText    51    51    51    )
( display  winColor1  166  166  166    )
( display  winColor2  115  115  115    )
( display  winColor3  189  204  204    )
( display  winColor4  204  204  204    )
( display  winColor5  199  199  199    )
)

derDefineStipple(
    ;( DisplayName  StippleName  Bitmap )
    ( display  blank         (  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
        (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)  
    )
    )
    ( display  solid       (  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
        (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)  
    )
)
Technology File and Display Resource File User Guide
Technology File and Display Resource File Examples

(1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
(1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
(1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
(1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
(1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
(1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
(1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
)

( display dots )
( 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 )
( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
( 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 )
( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
( 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 )
( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
( 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 )
( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
( 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 )
( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
( 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 )
( 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )
)

( display x )
( 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 )
( 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 )
( 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 )
( 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 )
( 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 )
( 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 )
( 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 )
( 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 )
( 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 )
( 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 )
( 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 )
( 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 )

April 2001 363 Product Version 4.4.6
Technology File and Display Resource File User Guide
Technology File and Display Resource File Examples

(1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0)
(0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1)
(0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0)
(0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1)
)
)

drDefineLineStyle(

;;( DisplayName   LineStyle     Size    Pattern )
( display       solid          1      (1 1 1) )
( display       dashed         1      (1 1 1 0 0) )
( display       dots           1      (1 0 0) )
( display       dashDot        1      (1 1 0 0 1 0 0) )
( display       shortDash      1      (1 1 0 0) )
( display       doubleDash     1      (1 1 1 0 0 1 1 0 0) )
( display       hidden         1      (1 0 0 0) )
( display       thickLine      3      (1 1 1) )
( display       bigDash        2      (1 1 1 0 0 1 1 1 0 0) )
)

drDefinePacket(

;;( DisplayName PacketName Stipple LineStyle Fill Outline )
( display blacksolid_S solid solid black black )
( display blue blank solid blue blue )
( display bluedashed_L blank dashed blue blue )
( display bluevZigZag_S vZigZag solid blue blue )
( display browndashed_L blank dashed brown brown )
( display creamsolid_S solid solid cream cream )
( display cyan blank solid cyan cyan )
.
.
.
( display yellow blank solid yellow yellow )
( display yellowX_SB X solid yellow yellowB )
( display yellowthickLine_L blank thickLine yellow yellow )
)

drDefinePacketAlias( "psb" "metall" "bluevZigZag_S"")
drDefinePacketAlias( "psb" "metal2" "bluevZigZag_S"")
drDefinePacketAlias( "psb" "net" "bluevZigZag_S"")
drDefinePacketAlias( "psb" "net2" "bluevZigZag_S"")