Technology File and Display Resource File User Guide

Product Version 5.0
January 2003

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Contents

A
Form Descriptions

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

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 reference information about ASCII syntax for the technology file and the display resource file, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.
- For reference information about the DFII SKILL API for the technology file, refer to the *Technology File and Display Resource File SKILL Reference Manual*.
- 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 the Diva® verification product rules files, see the *Diva 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 the Virtuoso XL Layout Editor (Virtuoso XL), see "Editing Your Technology File for Virtuoso XL Layout Editor" in the *Virtuoso XL Layout Editor 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 information about the differences between LEF technology data and the technology file for the Preview Silicon Ensemble™ place-and-route software, see the *Design Data Translator's Reference*, the LEF Data Map section of Chapter 7.

Typographic and Syntax Conventions

The following sections explain the typographic and syntax conventions used in this document.

Syntax Conventions

This list describes the syntax conventions used in this document.

`text`	Indicates text you must type exactly as it is presented.
`z_argument`	Indicates text that you must replace with an appropriate argument. The prefix (in this case, *`z_`*) indicates the data type the argument can accept. Do not type the data type or underscore.
`[ ]`	Denotes optional arguments. When used with vertical bars, they enclose a list of choices from which you can choose one.
`{ }`	Used with vertical bars and encloses a list of choices from which you must choose one.
`|`	Separates a choice of options.
…	Indicates that you can repeat the previous argument.
=>	Precedes the values returned by a Cadence® SKILL language function.
/	Separates the possible values that can be returned by a Cadence SKILL language function.
text	Indicates names of manuals, menu commands, form buttons, and form fields.

The language requires many characters not included in the preceding list. You must type these characters exactly as they are shown in the syntax.

Data Types

The Cadence® SKILL language and ASCII file syntax support several data types to identify the type of value you can assign to an argument. Data types are identified by a single letter followed by an underscore; for example, *t* is the data type in *t_viewNames*. Data types and the underscore are used as identifiers only: they are not to be typed.

The table below lists all data types supported by SKILL.

Prefix	Internal Name	Data Type
a	array	array
b	ddUserType	Boolean
c	opfcontext	OPF context
d	dbobject	Cadence database object (CDBA)
e	envobj	environment
f	flonum	floating-point number
F	opffile	OPF file ID
g	general	any data type
G	gdmSpecIlUserType	gdm spec
h	hdbobject	hierarchical database configuration object
l	list	linked list
m	nmpIlUserType	nmpIl user type
M	cdsEvalObject	—
n	number	integer or floating-point number
o	userType	user-defined type (other)
p	port	I/O port
q	gdmspecListIlUserType	gdm spec list
r	defstruct	defstruct
R	rodObj	relative object design (ROD) object
s	symbol	symbol
S	stringSymbol	symbol or character string
t	string	character string (text)

Prefix	Internal Name	Data Type
u	function	function object, either the name of a function (symbol) or a lambda function body (list)
U	funobj	function object
v	hdbpath	—
w	wtype	window type
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

ASCII and DFII SKILL Syntax Examples

The following examples show typical syntax characters used in the technology file and display resource file ASCII syntax and DFII 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.

l_result	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.

1

About the Technology File and Display Resource File

The chapter discusses the following:

Cadence Design Technology Data

The Cadence® design technology data defines the parameters used in design sessions. The Cadence design 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 Cadence design technology data is distributed in two types of files, the technology file and the display resource file, as described in this User Guide. Some products (Diva® verification products, for example) 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
- Place and route 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 the Diva verification products. For more information, refer to the *Diva Reference*.

How Technology Data Fits into and Is Used in the Design Flow

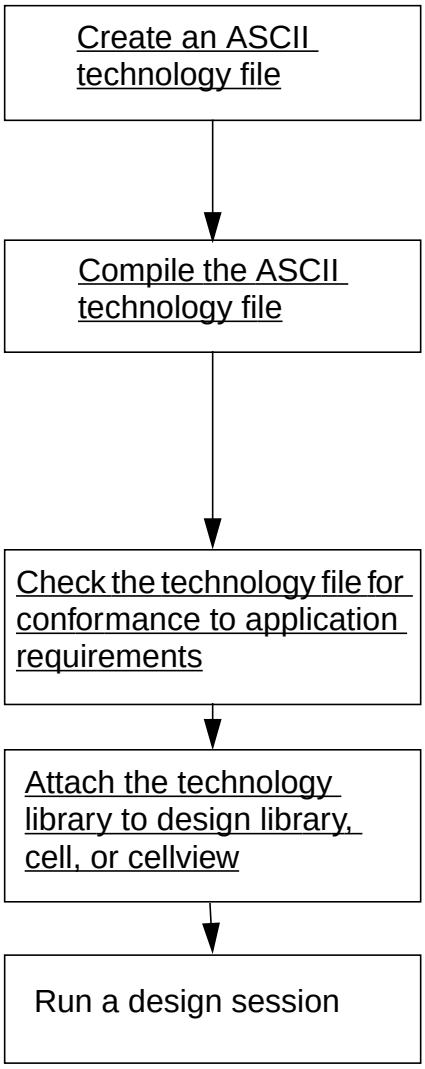
To run a Cadence design session, you must define technology data in one or more technology files and display resource data in one or more display resource files. You then compile the technology file or files to create a technology library, which contains a binary technology file and device cellviews 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:



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 contains technology data, commented as you see fit. 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.

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**.**

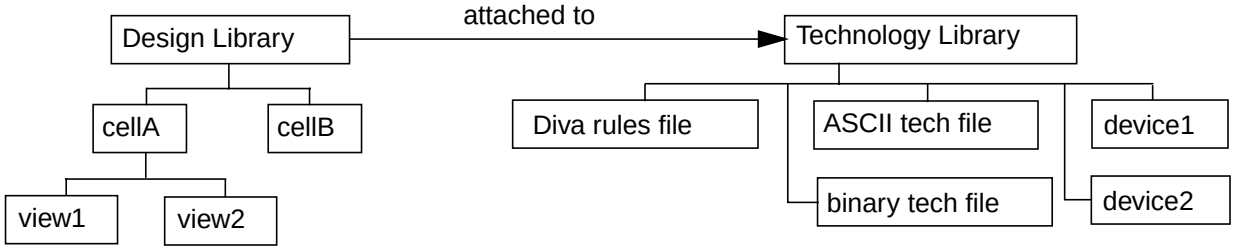
Checking ensures that the technology file data conforms to the requirements of the Cadence design applications with which it is to be used.

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.

When you open a design library, the application 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 Technology File Tool Box commands and DFII SKILL functions to set and retrieve technology data.

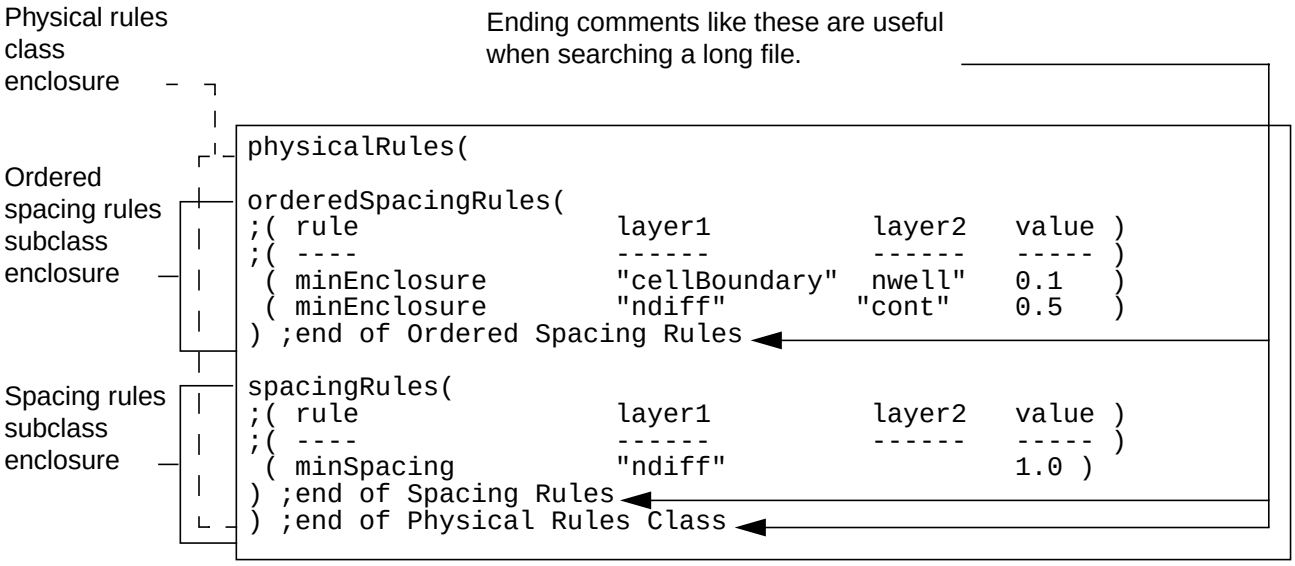
Every design uses a technology library. Usually, all of the designs in a design library use the same technology library, but Cadence applications do not require that you follow this model. You can use any number of technology libraries in your design hierarchy; their number and contents depend upon your 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.



Technology File Organization

The technology file is organized into classes and subclasses. 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.



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 46.

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 50.

The Devices class (`devices`) defines the devices you use with specific Design Framework II (DFII) applications. 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
  - 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 (Design Framework II Only)" on page 59.

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`)
- Define layer functions for layers and layer-purpose pairs (`layerFunctions`)

For more information about layer rules, refer to "Layer Rules" on page 83.

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

- List spacing rules in which the order of layers is not important (`spacingRules`)
- List spacing rules in which the order of layers is important (`orderedSpacingRules`)
- Define lookup tables for determining spacing rules based on specified conditions (`tableSpacingRules`)
- Specify a multiple for grid snapping (`mfgGridResolution`)

For more information about creating physical rules, refer to "Physical Rules" on page 90.

The Electrical Rules class (`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 (`characterizationRules`)
- List characterization rules in which the order of layers is important (`orderedCharacterizationRules`)
- Define lookup tables for determining characterization rules based on specified conditions (`tableCharacterizationRules`)

For more information about creating electrical rules, refer to "Electrical Rules" on page 101.

Application-specific rules classes apply only to specific design applications. They define rules that apply to specific Cadence physical design applications (Virtuoso Layout Editor, Virtuoso XL Layout Editor, and Virtuoso Compactor) and place-and-route applications (such as Preview Silicon Ensemble™).

The Layout Editor Rules class (`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) (`leLswLayers`)

For more information about specifying layout editor rules, refer to "Virtuoso Layout Editor Rules" on page 110.

The Virtuoso XL Rules class (`lxRules`) specifies rules for the Virtuoso XL Layout Editor (Virtuoso XL) physical design application. The individual subclasses

- List the layers to be monitored by the online extractor (`lxExtractLayers`)
- List the layers that cannot overlap in a Virtuoso XL design (`lxNoOverlapLayers`)
- Define templates for relative object design (ROD) multipart paths (MPPs) (`lxMPPTemplates`)

For more information about specifying Virtuoso XL rules, refer to "Virtuoso XL Layout Editor Rules" on page 113.

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 117.

The Place and Route Rules class (`prRules`) specifies rules for place- and-route applications. 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 the maximum number of stacked vias allowed within a specified layer range (`prMaxStackVias`)
- Define master slice layers (`prMastersliceLayers`)
- Define the rules for placing vias and turn vias (`prViaRules`)
- Define the rules for generating vias and turn vias (`prGenViaRules`)
- Define the rules for routing with nondefault wire widths (`prNonDefaultRules`)
- Specify minimum allowable spacing between two regular geometries on different nets (`prRoutingPitch`)
- Define the distance between the placement grid and the routing grid when there is a routing grid between two placement grids (`prRoutingOffset`)
- Define the overlap layer or layers used to display the overlap boundary (`prOverlapLayer`)

For more information, refer to “Place and Route Rules” on page 126.

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 of the technology file are used by which Cadence applications, as identified in the table by the following numbers:

1. Virtuoso® Layout Editor
2. Virtuoso XL Layout Editor
3. Virtuoso Compactor
4. Preview Silicon Ensemble™
5. Preview Gate Ensemble®
6. Other applications

Technology File Class or Subclass	Application					
	1	**2**	**3**	**4**	**5**	**6**
controls()						
techParams() techPermissions()	The Controls class sets parameters that can be used throughout the technology file.					
layerDefinitions()						
techLayers()	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()	Layer properties are user defined.					
devices()						
tcCreateCDSDeviceClass()	x	x	x			
symContactDevice()	x	x	x			
ruleContactDevice()						x

Technology File Class or Subclass	Application					
	1	2	3	4	5	6
symEnhancementDevice()			x			
symDepletionDevice()			x			
symPinDevice()	x	x	x			
symRectPinDevice()	x	x	x			
cdsViaDevice()	x	x	x			
tcCreateDeviceClass()		x	x			
tcDeclareDevice()		x	x			
layerRules()						
viaLayers()	x	x		x		
equivalentLayers()		x	x			
streamLayers()	To translate designs with the pipo translator, use `streamLayers`.					
layerFunctions()	x	x	x	x	x	x
physicalRules()						
spacingRules()	x		x	x	x	
orderedSpacingRules()			x	x	x	x
tableSpacingRules()				x	x	x
mfgGridResolution()	x	x	x	x	x	x
electricalRules()						
orderedCharacterizationRules()		x	x	x	x	x
characterizationRules()		x	x	x	x	x
tableCharacterizationRules()				x	x	x
leRules()						
leLswLayers()	x	x	x			
lxRules()						

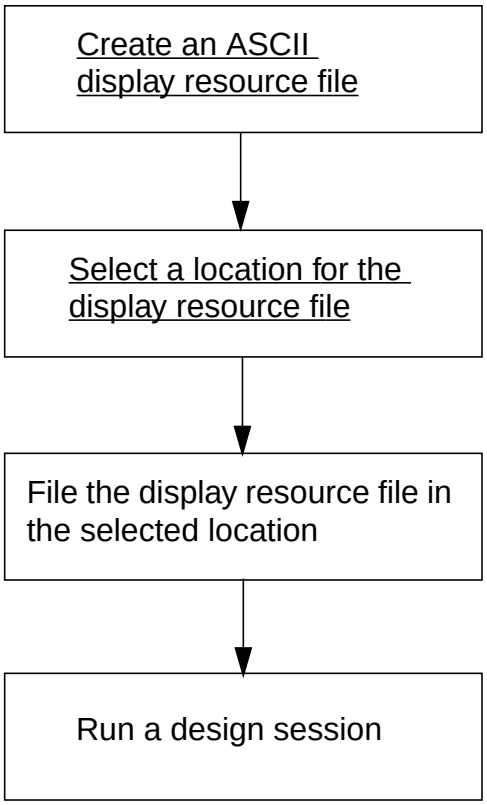
Technology File Class or Subclass	Application					
	1	2	3	4	5	6
lxExtractLayers()		x				
lxNoOverlapLayers()		x				
lxMPPTemplates()		x				
compactorRules()						
compactorLayers()			x			
symWires()			x			
symRules()			x			
prRules()						
prRoutingLayers()				x	x	
prViaTypes()				x	x	
prStackVias()				x	x	
prMaxStackVias()				x	x	
prMastersliceLayers()				x	x	
prViaRules()				x	x	
prGenViaRules()				x	x	
prNonDefaultRules()				x	x	
prRoutingPitch()				x	x	
prRoutingOffset()				x	x	
prOverlapLayer()				x	x	

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:



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`.

When selecting where to file your `display.drf` file, you must consider that Cadence design software loads and merges up to six display resource files from predefined locations. See "How Cadence Design Software Handles Multiple Display Resource Files" on page 33 for details.

As mentioned above, when you open a design library, Cadence design 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 DFII SKILL functions.

How Cadence Design Software Handles Multiple Display Resource Files

Cadence design 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

  `your_install_dir/share/cdssetup/dfII/default.drf`

  This file is used with the Virtuoso Schematic Composer.

- 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 editor. For further information, refer to Chapter 12, "Editing, Reusing, and Merging Display Resources."

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 157.

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 158.

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 159.

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 160.

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 161.

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 164.

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:

1. First, in the technology file, the design software finds the name of the display packet assigned to the layer.
2. 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.

The following 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

After defining layers and purposes, defines layer-purpose pairs and assigns display packets (plus attributes not shown here).

```
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

Defines the display devices.

Then, for each, defines the following:

colors

stipples

line styles

display packets

```
.
drDefineDisplay(
;(DisplayName  #Colors  #Stipple  #LineStyles  )
 (display      52        32        32           )
...)
drDefineColor(
;(DisplayName   ColorsName   Red      Green    Blue)
 (display       yel          255      255      0   )
 (display       silver       217      230      255 )
...)
drDefineStipple(
;(DisplayName StippleName  Bitmap     )
 (display     dots         ( ( 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0)
...)
drDefineLineStyle(
;(DisplayName   LineStyle    Size       Pattern   )
 (display       solid        1          (1 1 1) )
...)
drDefinePacket(
;(DisplayName PacketName      Stipple  LineStyle  Fill  Outline)
 ( display    yelsilverdots_S dots     solid      yel   silver )
...)
...
```

The Technology File Manager and User Interface

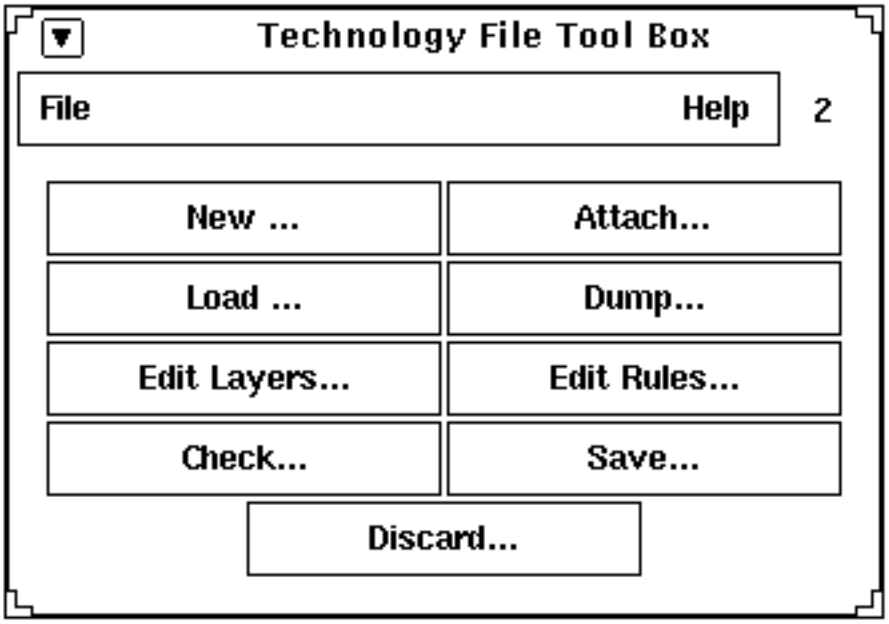
The technology file manager provides a user interface that allows you to manipulate technology data in virtual memory during a design session.

Invoking the Technology File Manager

To start the technology file manager, enter the following command on the UNIX command line, prefaced by the path to the directory containing the Cadence tools:

```
techManager &
```

The technology file manager displays the Technology File Tool Box.



SKILL Functions to Display Form (DFII Only)

```
techManagerOpenTechToolBox()
```

This SKILL function displays the Technology File Tool Box. It is equivalent to the CIW *Tools – Technology File Manager* command.

```
techManagerToolBox()
```

This SKILL function displays the Technology File Tool Box. It is equivalent to the CIW *Tools – Technology File Manager* command.

Technology File Tool Box Commands

The Technology File Tool Box commands let you compile, dump, and edit technology data. This section introduces the Technology File Tool Box commands. For detailed information about using these commands, refer to "Editing Class Data through the Technology File Tool Box" on page 191.

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.

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

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.

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

Edit Rules lets you edit various technology file classes and subclasses.

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.

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

The Display Resources Manager and User Interface

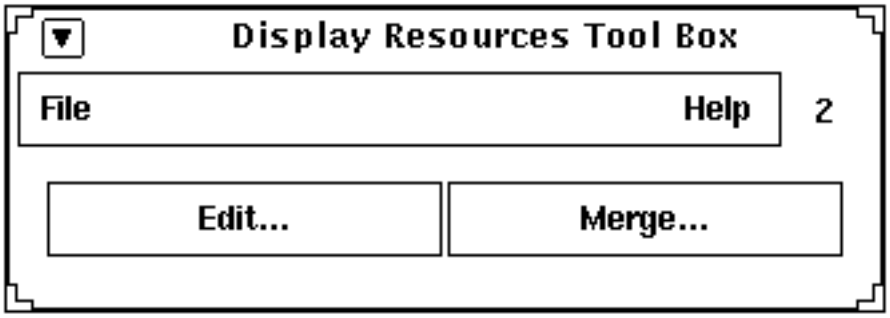
The display resources manager provides a user interface that allows you to manipulate display resources in virtual memory during a design session.

Invoking the Display Resources Manager

To start the display resources manager, enter the following command on the UNIX command line, prefaced by the path to the directory containing the Cadence tools:

```
displayManager &
```

The display resources manager displays the Display Resources Tool Box.



SKILL Function to Display Form (DFII Only)

```
techManagerOpenDisplayToolBox()
```

This SKILL function displays the Technology File Tool Box. It is equivalent to the CIW *Tools – Display Resource Manager* command.

Display Resources Tool Box Commands

The Display Resources Tool Box commands let you invoke the Display Resource Editor and merge display resource files. This section introduces the Display Resources Tool Box commands. For detailed information about using these commands, refer to Chapter 12, "Editing, Reusing, and Merging Display Resources."

Edit invokes the Display Resource Editor. 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.

Merge 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*.

2

Creating a Technology File: Methods and General Guidelines

This chapter discusses the following:

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 Virtuoso® Layout Editor Rules, Virtuoso XL Rules, Virtuoso Compactor 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 any user-defined layer purpose, properties, or rules required by any application or software you are using in your design flow in appropriate places in the technology file. Note, however, that specifying a user-defined rule in the technology file does not force all applications to recognize that rule; if a user-defined rule is not used by a specific tool, then that tool will simply ignore the rule.

- 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. For a complete list of data types supported in the ASCII syntax and by DFII SKILL, see the *Preface*.

Technology File Statements

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

- `include`
- `comment`

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 file `devices.def` 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:

```
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.

3

Creating a Technology File: Specifying Controls, Layer Definitions, and Device Definitions

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

- Controls on page 46
- Layer Definitions on page 50
- Devices (Design Framework II Only) on page 59

Other technology file classes define the following:

- Layer rules, described in Chapter 4, “Creating a Technology File: Specifying Generic Rules.”
- Physical rules, described in Chapter 4, “Creating a Technology File: Specifying Generic Rules.”
- Electrical rules, described in Chapter 4, “Creating a Technology File: Specifying Generic Rules.”
- Application-specific rules, described in Chapter 5, “Creating a Technology File: Specifying 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, along with the technology file controls they define.

```
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") )
              .
              .
              .
    ( prRules    ("jrm")          ("lmh") )

    ) ; end of techPermissions

) ; end of controls
```

Beginning of Controls class enclosure. All subclasses must be specified within the parentheses of the class enclosure.

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.

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

For more information about the `controls` class, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

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 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:

```
techParam ( paramName )
```

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

```
 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 `techParam("theta") * 2` as the rule value for the `minWidth` spacing rule and the expression `techParam("theta")` as the rule value for the `currentDensity` characterization rule; it evaluates each expression when it is accessed, as follows:

```
minWidth = 2.0 * 2 = 4.0
currentDensity = 2.0
```

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 from the Technology File Tools Box by clicking on *Edit Rules* and then accessing the *Technology File – Control* form
- In virtual memory during a design session with the `techSetParam` SKILL function (DFII only)

For more information about the `techParams()` subclass, refer to the *Technology File and Display Resource File ASCII Syntax 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, all users have read/write access to that class.

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

Technology Permissions 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") )
                .
            .
                .
  ( prRules    ("jim")          ("lmh") )

   ) ; end of techPermissions
```

For more information about the `techPermissions()` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Layer Definitions

In Cadence® designs, a 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 designs.

Sample Layer Definitions Class

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

```
layerDefinitions(
```

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

```
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
```

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.

```
techPurposes(
    ;( PurposeName    Purpose#  [Abbreviation] )

    ;User-Defined Purposes:
    ;System-Reserved Purposes:

     ( warning        234        wng           )
     ( tool1          235         tl1          )
              .
              .
              .
     ( cell           254         cel          )

      ) ;end of techPurposes
```

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.

```
techLayerPurposePriorities(
    ;layers are ordered from lowest to
    ;highest priority
    ;( layerName              Purpose          )

     ( background             drawing          )
     ( grid                   drawing          )
              .
              .
              .
     ( Unrouted               drawing9         )

      ) ;end of techLayerPurposePriorities
```

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.

```
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 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 an Diva® verification products 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.

```
techLayerProperties(
    ;( PropName     Layer1   [Layer2]  PropValue )

     ( defaultWidth  ndiff             1.000000  )
               .
               .
               .
     ( defaultWidth   pdiff             1.000000 )
      ) ;end of techLayerProperties
```

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.

```
) ;end of layerDefinitions
```

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

For more information about the `layerDefinitions` class, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

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: `metal1`, `poly`, `ndiff`). Cadence applications also use special layers to display warnings and for highlighting. You need to create the layers that are specific to your designs.

Cadence design software 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 of the Layer Definitions class defines the layers to be used in your design.

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).

```
techLayers(

     ;( layerName     Layer#   [Abbreviation] )
      ( ndiff          1                      )
      ( pdiff          2                      )
      ( pwell         13                      )
      ( cellBoundary  16        clBndry       )
      ( nimplant      39        nmplant       )
      ( pimplant      40        pmplant       )

 ) ; end of techLayers
```

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()` subclass, refer to the *Technology File and Display Resource File ASCII Syntax 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.

Cadence design software 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 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()` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

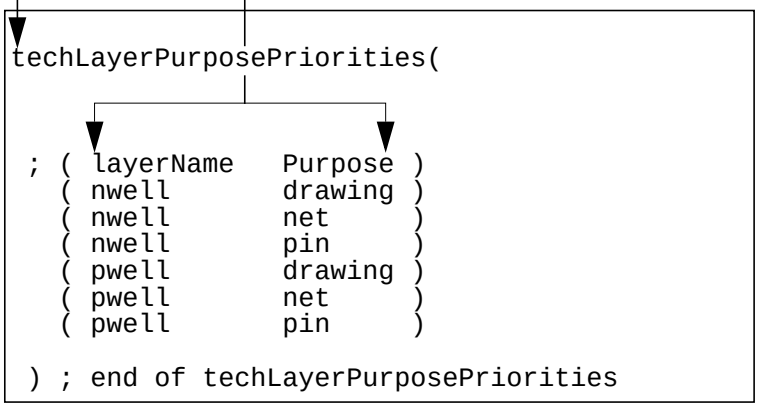
Defining Layer Display Priorities: techLayerPurposePriorities

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 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     )

 ) ; end of techLayerPurposePriorities
```

For more information about the `techLayerPurposePriorities()` subclass, refer to the *Technology File and Display Resource File ASCII Syntax 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 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 verification software tracks changes to objects in incremental verification.

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

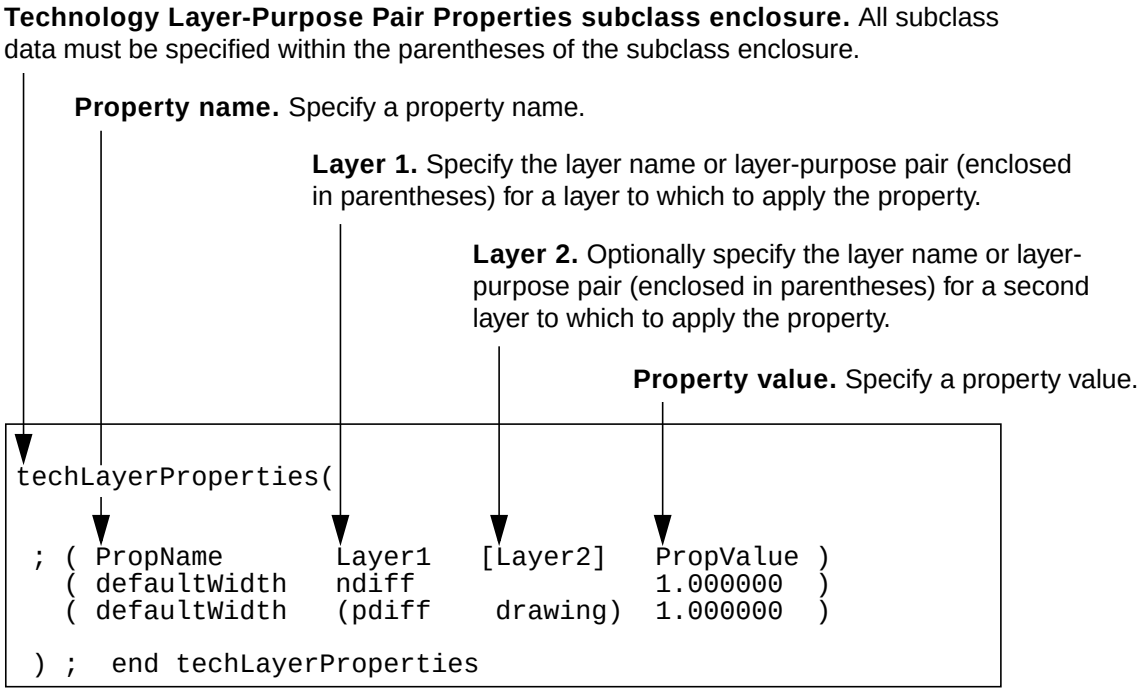
Valid. When set to `t`, the layer-purpose pair is displayed on the LSW.

```
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     t        nil    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()` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying User-Defined Layer Properties: techLayerProperties()

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



For more information about the `techLayerProperties()` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Devices (Design Framework II Only)

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-layer or double-layer
- rectPin — Rectangular dot pin
- via — Via

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

your_install_dir`/samples/techfile/devices.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
  - Cadence DFII 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® XL Layout Editor (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:

- "Editing Your Technology File for Virtuoso XL Layout Editor" in the *Virtuoso XL Layout Editor User Guide*
- "Getting Started" in the *Virtuoso Compactor Reference Manual*
- "Translating LEF and DEF Files" in the *Design Data Translator's Reference*
- "LEF Data Map" in the *Design Data Translator's Reference*

Sample Devices Class

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

For more information about the `devices` class, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

```
devices(
```

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

```
tcCreateCDSDeviceClass()
```

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

```
 symContactDevice(
;( deviceName   viaLayer   viaPurpose
 ( M1_P         cont       drawing

;  layer1   purpose1   [implant1]
   diff     drawing    (pimplant drawing 0.3)


;  layer2   purpose2   [implant2]
   metal1   drawing

;  width  length  [( row  column  xPitch  yPitch  xBias  yBias )]
   .6     .6       ( 1    1       _NA_    _NA_    _NA_   _NA_  )

;  encLayer1   encLayer2   legalRegion                )
   .6          .6          ( outside nwell drawing ) )
               .
               .
               .
 ) ;end of symContactDevice
```

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.

```
 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 )]
  ( metal1   drawing    ( -21.000  -21.000  21.000  21.000 ) )
               .
               .
               .
 ) ;end of ruleContactDevice
```

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
  poly1       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
  poly1       drawing

; width  length  sdExt  gateExt  legalRegion                 )
  1.8    0.6     1.2    0.9      ( inside  pwell  drawing ) )
               .
               .
               .
 ) ;end of symEnhancementDevice
```

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.

```
 symDepletionDevice(

;( deviceName   sdLayer   sdPurpose
 ( D_NTR        diff      drawing

; [ implantEnclosure ]


; gateLayer   gatePurpose   deplLayer   deplPurpose
  poly1       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 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.

```
 symPinDevice(

;( deviceName   maskable
 ( poly1_T      t

; layer1   purpose1   width1
  poly1    drawing    0.6

;[ layer2   purpose2   width2 ]   [ legalRegion ] )
                                                  )
               .
               .
               .
 ) ;end of symPinDevice
```

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.

```
symRectPinDevice(

;( deviceName   maskable
 ( nplus_P      nil

; layer1   purpose1   width   length   [ legalRegion ] )
  poly1    drawing    1       2                        )
               .
               .
               .
 ) ;end of symRectPinDevice
```

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.

```
cdsViaDevice(

;( deviceName  cutLayer  cutPurpose
 ( m1_m2       via1      drawing

;  layer1  purpose1  layer2  purpose2
   metal1  drawing   metal2  drawing

;  row   column   origin        stackedVias
   1     1        centerCenter  _NA_

;  cutLayerW  cutLayerL  xCutSpacing  yCutSpacing
   _NA_       _NA_       _NA_         _NA_

;  layer2XEnc  layer1YEnc  layer2XEnc  layer2YEnc
   _NA_        _NA_        _NA_        _NA_

;  layer1Dir  layer2Dir )
   ""         ""        )
) ; end of cdsViaDevice
```

Declare Via Device subclass. Declares via devices. Data for each via device must be enclosed in parentheses. All via 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 = 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)
;
 ) ;end of tcCreateDeviceClass
```

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.

```
 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
```

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
```

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 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()` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Declaring a Contact Device between Two Layers: symContactDevice()

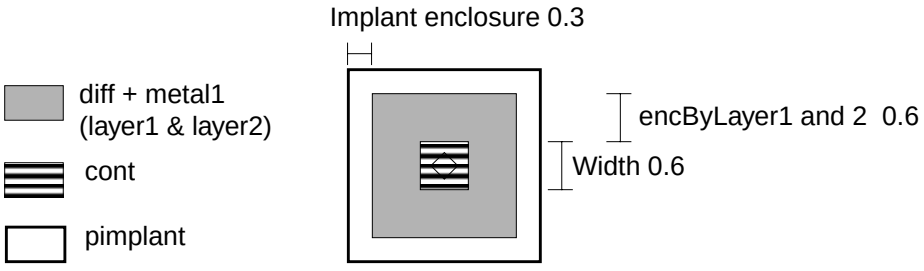
A contact device is a single contact or rectangular array of contacts placed between two layers. The layers overlap the contacts uniformly and a dot pin is placed in the center of each contact.

The `symContactDevice()` subclass of the Devices class declares contact devices. The following example defines a single contact named `M1_P`:

```
symContactDevice(
 ( deviceName  viaLayer  viaPurpose  layer1  purpose1  [implant1]
 ( M1_P         cont      drawing     diff    drawing    (pimplant drawing 0.3)

; layer2  purpose2  width  length  [( row column xPitch yPitch xBias yBias )]
  metal1  drawing   .6     .6       ( 1   1      _NA_   _NA_   _NA_  _NA_  )
)

 ; encLayer1   encLayer2   legalRegion )
   .6           .6          (outside nwell drawing) )

) ; end of symContactDevice
```

contact name
via layer name and purpose
layer1 name and purpose
optional enclosure layer and spacing (available for layer2 as well)
layer2
width
length
number of rows and columns for contact array
minimum allowable space, center-to-center, between contacts in x and y direction for contact array
orientation of origin for contact array
space between via and layer2
specifies whether contacts must be inside or outside the specified layer
space between via and layer1

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` 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.

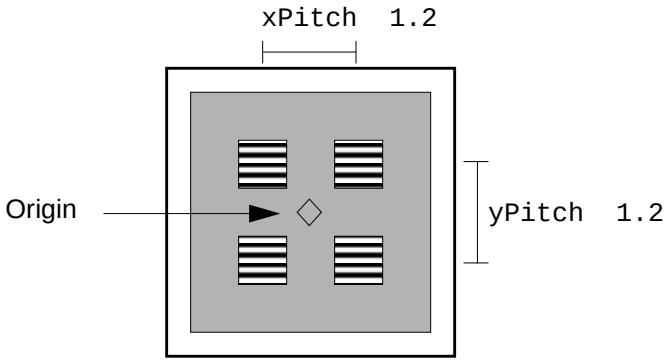
The `xBias` and `yBias` values determine where the array origin is placed. Possible values are calculated as follows:

- `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:



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` subclasses of the technology file.

The `ruleContactDevice()` subclass of the Devices class declares rule contact devices. The following example defines a rule contact named `VIA12`:

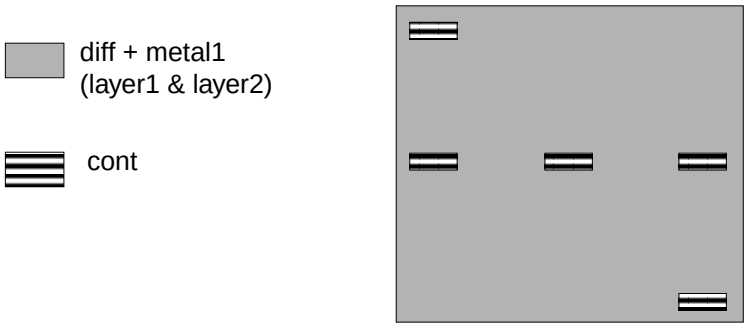
```
ruleContactDevice(
 ; ( deviceName
   ( "VIA12"
 ;   ( layer1   purpose1   rectangles
     ( diff     drawing    ( -21.000  -21.000   21.000   21.000 ) )
     ( cont     drawing    (  -2.400   -0.800    2.400    0.800 )
                           ( -19.000   17.400  -14.200   19.000 )
                           (  14.200  -19.000   19.000  -17.400 )
                           ( -19.000   -0.800  -14.200    0.800 )
                           (  14.200   -0.800   19.000    0.800 ) )
     ( metal1   drawing    ( -21.000  -21.000   21.000   21.000 ) )
)
```

the bottom, via, and top layers making up the rule contact device

coordinates of the rectangles of each layer

```
) ; end of ruleContactDevice
```

The rule contact appears as follows:



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 of the Devices class declares enhancement devices. The following examples define a PTR device and an NTR device.

```
symEnhancementDevice(
;( deviceName   sdLayer   sdPurpose  [ implantEnclosure ]
 ( PTR          diff      drawing    ( pimplant  drawing  0.3 )
```

optional enclosure layer, purpose, and spacing

```
; gateLayer   gatePurpose
  poly1       drawing
```

gate layer and purpose

```
; width   length   sdExt   gateExt   legalRegion                   )
  1.8     0.6      1.2     0.9       ( outside  pwell  drawing ) )
```

length

width

```
;( deviceName   sdLayer   sdPurpose [ implantEnclosure ]
 ( NTR          diff      drawing
```

source/drain layer and purpose

```
; gateLayer   gatePurpose
  poly1       drawing

; width   length   sdExt   gateExt   legalRegion                  )
  1.8     0.6      1.2     0.9       ( inside  pwell  drawing ) )
```

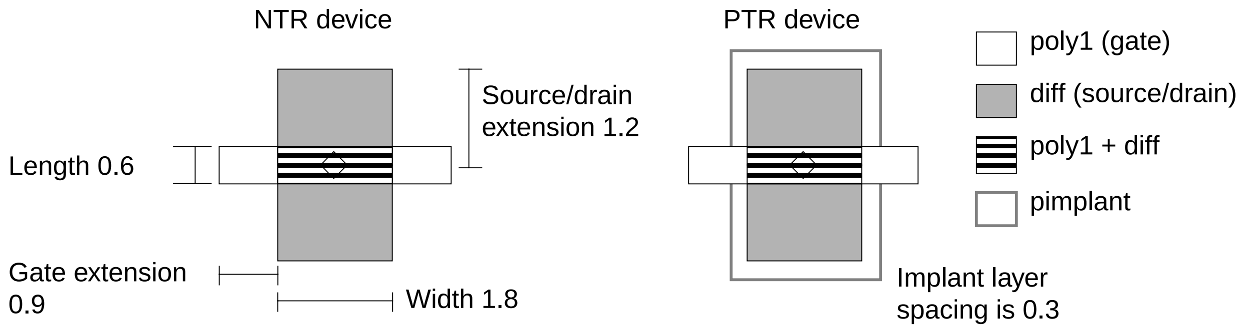
source/drain extension

gate extension

specifies whether devices must be inside or outside the specified well layer

```
;) end of symEnhancementDevice
```

The devices appear as follows:



Declaring a Depletion Device: symDepletionDevice()

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

The `symDepletionDevice()` subclass 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.

```
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
```

width

length

source/drain extension

gate extension

spacing of depletion enclosure around source/ drain

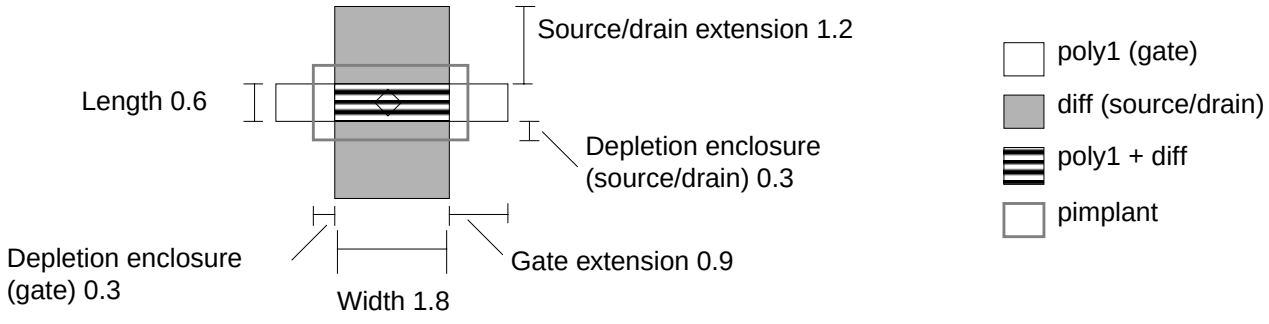
spacing of depletion enclosure around gate

```
; [ legalRegion ]              )
  ( outside  pwell  drawing ) )
```

specifies whether devices must be inside or outside the listed well layer

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

```
symPinDevice(
; ( deviceName   maskable   layer1   purpose1   width1
  ( poly1_T      nil         poly1    drawing   0.6

; [ layer2   purpose2   width2 ]   [ legalRegion ] )
    _NA_     _NA_       _NA_          _NA_          )

) ; end of symPinDevice
```

pin name

specifies whether pin is maskable

pin layer1

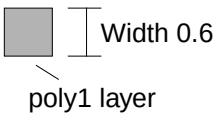
layer1 width

pin layer2

layer2 width

specifies whether pins must be inside or outside the listed well layer

The pin appears as follows:



Declaring a Rectangular Dot Pin Device: symRectPinDevice()

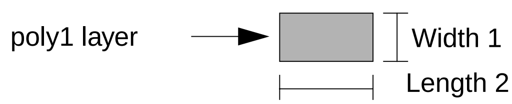
A rectangular pin device is a rectangular dot pin.

The `symRectPinDevice()` subclass of the Devices class declares rectangular pin devices. The following example declares a simple poly rectangular pin named `nplus_P`:

```
symRectPinDevice(
; ( deviceName   maskable   layer1   purpose1   width  length  [ legalRegion ] )
  ( nplus_P      nil        poly1    drawing    1      2         _NA_          )

) ; end of symRectPinDevice
```

- deviceName: **pin name**
- maskable: **sets whether this pin is maskable**
- layer1, purpose1: **pin layer and purpose**
- width: **width**
- length: **length**
- legalRegion: **specifies whether pins must be inside or outside the listed well layer**

The rectangular pin looks as follows:



Declaring a Via Device: cdsViaDevice()

A via device is a standard via.

The `cdsViaDevice()` subclass of the Devices class declares via devices. The following example declares a via named `m1_m2`:

```
cdsViaDevice(
;( deviceName  cutLayer  cutPurpose
 ( m1_m2       via1      drawing

;  layer1  purpose1  layer2  purpose2
   metal1  drawing   metal2  drawing

;  row   column         origin          stackedVias
   1     1              centerCenter    _NA_

;  cutLayerW  cutLayerL  xCutSpacing            yCutSpacing
   _NA_       _NA_       _NA_                   _NA_

;  layer2XEnc  layer1YEnc     layer2XEnc  layer2YEnc
   _NA_        _NA_           _NA_        _NA_

;  layer1Dir            layer2Dir )
   ""                   ""        )


) ; end of cdsViaDevice
```

via name

cut layer name and purpose

bottom routing layer name and purpose

top routing layer name and purpose

number of rows and columns of contact cuts in a contact array

origin of the contact or via cut

specifies whether vias can be stacked

width and length of the contact or via cut

distance between edge to edge of horizontal cuts in a contact array

distance between edge to edge of vertical cuts in a contact array

horizontal and vertical distance from the cut edge to the enclosure edge for the contact on layer 1

horizontal and vertical distance from the cut edge to the enclosure edge for the contact on layer 2

layer 1 direction(s) to apply to the X/Y enclosure

layer 2direction(s) to apply to the X/Y enclosure

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: tcCreateDeviceClass()

The `tcCreateDeviceClass()` subclass 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.

Note: Continued in following figures.

```
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

) ; end of tcCreateDeviceClass and syMGEnhancement device type definition
```

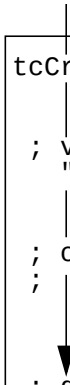
Class parameters. Class parameters are fixed once you declare a device. They usually define the physical characteristics of the device. Specify the parameter name followed by a default value for each parameter, enclosed in parentheses and separated by a space. You can specify any number of parameters 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.



```
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()` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Declaring User-Defined Devices: tcDeclareDevice()

The `tcDeclareDevice()` subclass 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 subclass—`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
```

Note: Continued in following figure.

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(

 ; view        deviceType          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()` subclass, refer to the *Technology File and Display Resource File ASCII Syntax 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 `tcSetDeviceClassProp()` subclass of the Devices class applies properties to a device type.

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 subclass—`tcCreateCDSDeviceClass()` or `tcCreateDeviceClass()`—that defines the device type.

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.

Device type. Specify the name of the device type 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.

```
tcSetDeviceClassProp(

  ; view        deviceType           (property    value )
    "symbolic"   "syMGDepletion"     ("function"  "transistor" )
         .
         .
         .
) ; end of tcSetDeviceClassProp
```

Applying Properties to an Individual Device: tcSetDeviceProp()

The `tcSetDeviceProp()` subclass 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 subclass 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
```

4

Creating a Technology File: Specifying 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 83
- Physical Rules on page 90
- Electrical Rules on page 101

Other technology file classes define the following:

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

Layer Rules

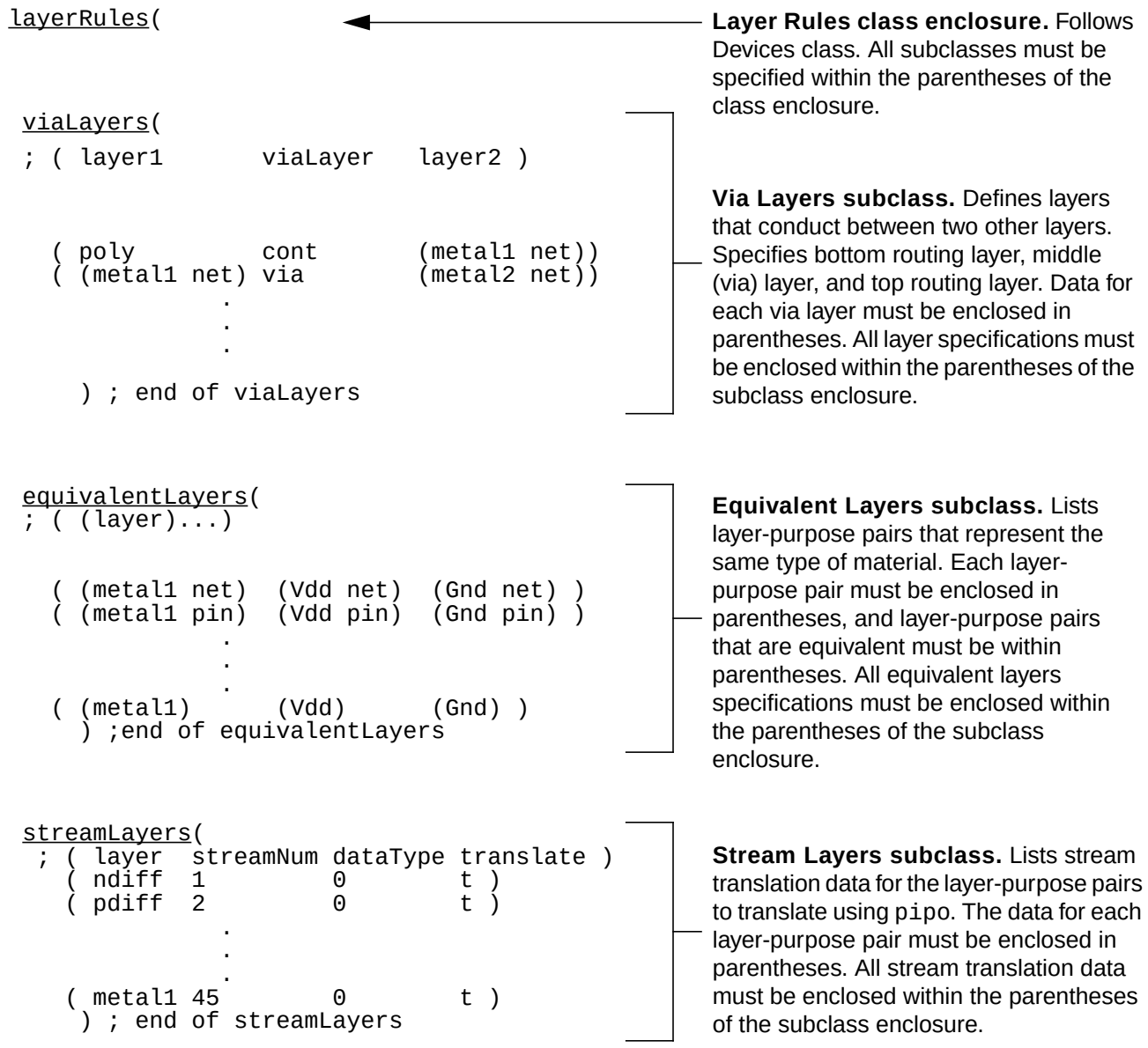
You specify layer rules to establish the relationships and interactions between layers. Layer rules define the following:

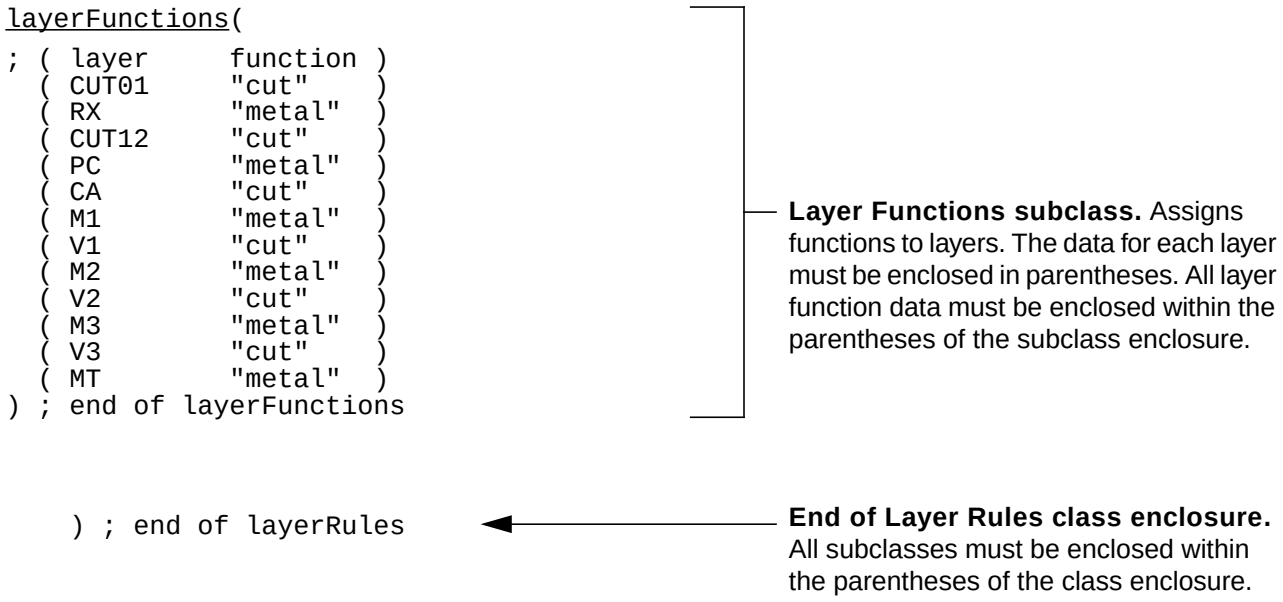
- Via layers that connect two conducting layers
- Layers that are physically and electrically equivalent
- Stream translation data for a layer

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, along with the layer rules they define.





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

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.

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
- Virtuoso XL Layout Editor (Virtuoso XL)
- Preview Silicon Ensemble™
- Preview Gate Ensemble®

The `viaLayers` subclass 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:

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).

Via layer. Specify the layer name or layer-purpose pair (enclosed in parentheses) for the middle layer (the via layer).

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` subclass, refer to the *Technology File and Display Resource File ASCII Syntax 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` subclass of the Layer Rules class of the technology file. Virtuoso XL uses technology file equivalent layers data.

The `equivalentLayers` subclass 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.

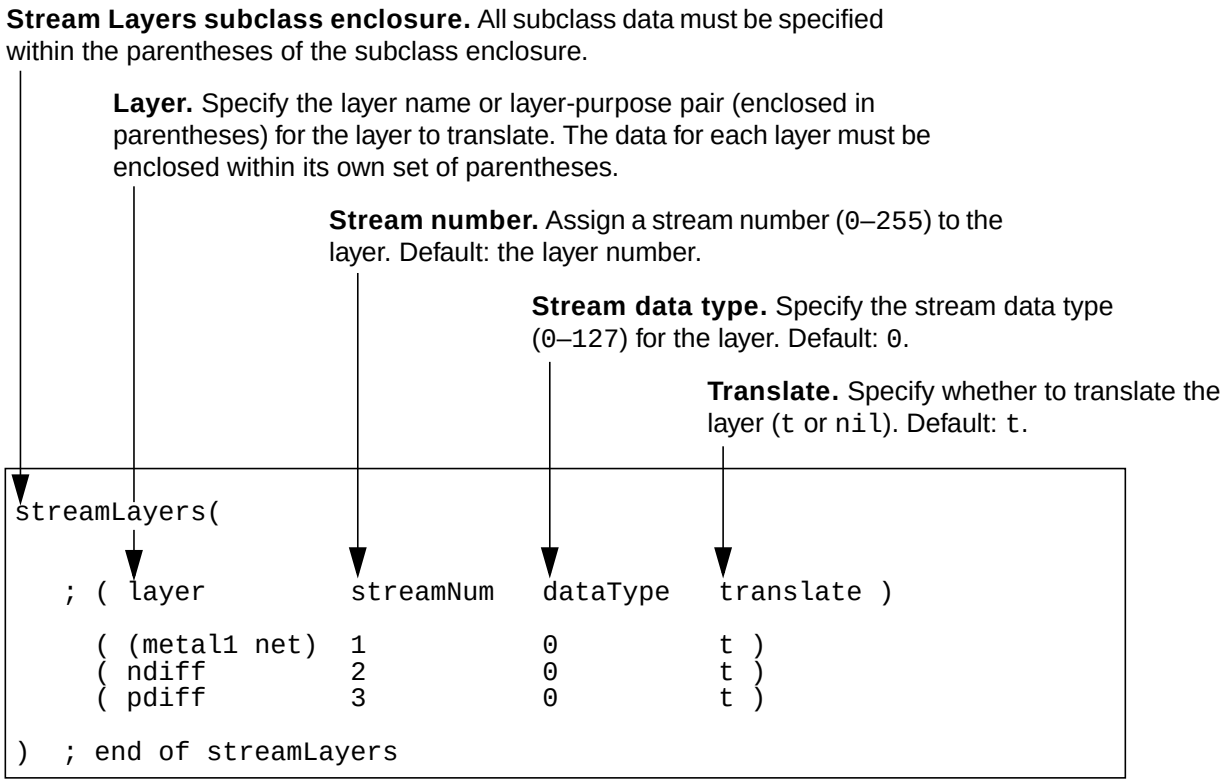
```
equivalentLayers(

   ; ( 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` subclass, refer to the *Technology File and Display Resource File ASCII Syntax 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 of the Layer Rules class specifies the stream layers for your design.



For more information about the `streamLayers` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

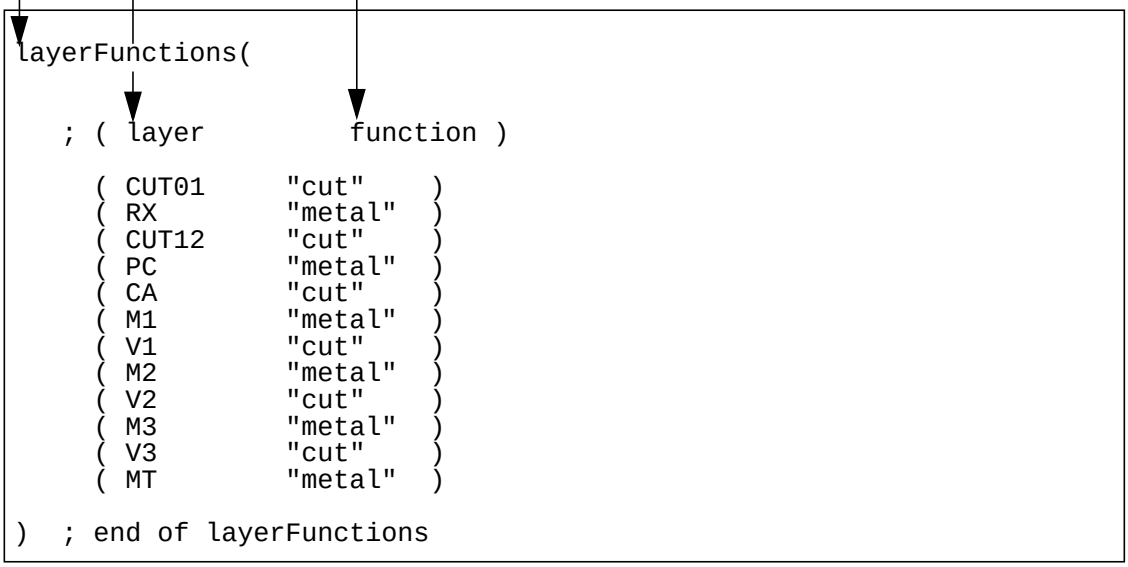
Defining Layer Functions: layerFunctions()

Layer functions define how a layer can be used. The layerFunctions subclass of the Layer Rules class assigns layer functions to the layers in your design.

Layer Functions 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 which to assign the function. The data for each layer must be enclosed within its own set of parentheses.

Function. Assign one of the following functions to the layer: `cut`, `li`, `metal`, `ndiff`, `nplus`, `nwell`, `pdiff`, `poly`, `pplus`, `pwell`



```
layerFunctions(

   ; ( layer          function )

     ( CUT01     "cut"    )
     ( RX        "metal"  )
     ( CUT12     "cut"    )
     ( PC        "metal"  )
     ( CA        "cut"    )
     ( M1        "metal"  )
     ( V1        "cut"    )
     ( M2        "metal"  )
     ( V2        "cut"    )
     ( M3        "metal"  )
     ( V3        "cut"    )
     ( MT        "metal"  )

)  ; end of layerFunctions
```

For more information about the `layerFunctions` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Physical Rules

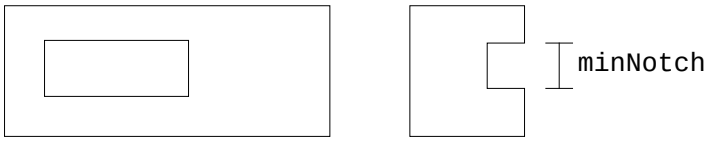
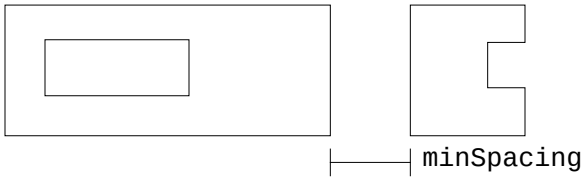
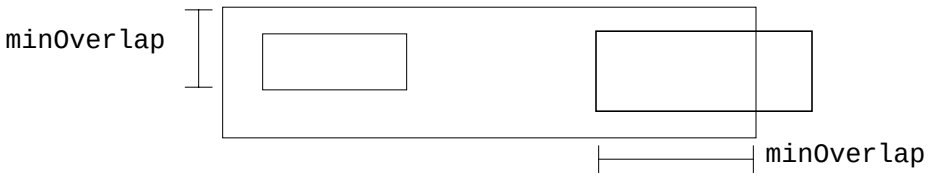
You specify physical rules to establish spacing within and between objects in your design and to specify the grid snapping. Physical rules are specified in the Physical Rules class, a generic rules class that can be used by all applications. Physical rules define design restrictions such as the following:

- 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

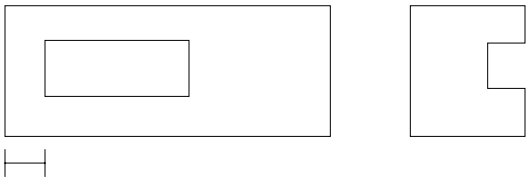
Spacing rules specify the distance required between layers and the width of objects and paths. Ordered spacing rules specify the distance required when one layer encloses another.

The following table shows rules recognized by Cadence design applications, the applications that use each rule, and the technology file subclass in which to define each rule. You can also specify any rule used by the design applications you employ, including any user-defined rules you use in your applications.

Type of Spacing Rule	Rule	Applications That Use It	Subclass of Physical Rules Class
Minimum width of a path on the specified layer	`minWidth`	Virtuoso Layout Editor, Virtuoso XL, Virtuoso Compactor, Preview Silicon Ensemble, Preview Gate Ensemble, Diva® verification products	`spacingRules`
Default width of a path on the specified layer	`defaultWidth`	Virtuoso Compactor	`spacingRules`
Maximum width of a path on the specified layer	`maxWidth`	Virtuoso Layout Editor, Virtuoso XL, Virtuoso Compactor	`spacingRules`

Type of Spacing Rule	Rule	Applications That Use It	Subclass of Physical Rules Class
Minimum distance between the outside facing edges of a notch drawn in an object	`minNotch`	Virtuoso Compactor, Virtuoso Layout Editor, Diva verification products	`spacingRules`
			
Distance between objects drawn on the specified layer or layers	`minSpacing`	Virtuoso Compactor, Preview Silicon Ensemble, Preview Gate Ensemble, Diva verification products	`spacingRules`
			
Minimum distance between the inside and outside edges of two overlapping objects	`minOverlap`	Diva verification products	`spacingRules`
			

Type of Spacing Rule	Rule	Applications That Use It	Subclass of Physical Rules Class
Distance by which an object must be enclosed by another object	`minEnclosure`	Compactor	`orderedSpacingRules`



Sample Physical Rules Class

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

```
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


 tableSpacingRules(
 ; ( rule             layer1        [layer2]
   ( "MINMIDWIRESPACE"  poly1          poly2

 ;   ( [index1Definitions]   [index2Definitions]
     (  "widthwireleft"       "widthwireright"

 ;    ( table ) ) )
      (
       ( 0.6 >=  0.6 >= )  0.35
       ( 0.6 <   0.6 >= )  0.25
       ( 0.6 >=  0.6 <  )  0.25
       ( 0.6 <   0.6 >  )  0.20
      )
     )
   )
 ) ; end of tableSpacingRules
```

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.

Spacing Rules Tables subclass. Defines spacing rules lookup tables. All table spacing rules must be enclosed within the parentheses of the subclass enclosure.

```
mfgGridResolution(
         ;( g_resolution )

           ( 0.001000 )
    ) ;end of mfgGridResolution
```

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 physicalRules
```

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

For more information about the `physicalRules` class, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

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.

Specifying Spacing Rules: spacingRules()

The `spacingRules` subclass 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 Physical Rules on page 90. 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     metal1            0.600000)
   ( minWidth     metal1            0.600000)
   ( minSpacing   metal1            0.600000)
   ( minSpacing   metal1   via      0.600000)
   ( defaultWidth metal1            0.600000)
   ( maxWidth     metal1            0.600000)
  )  ; end of spacingRules
```

For more information about the `spacingRules` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying Ordered Spacing Rules: orderedSpacingRules()

The `orderedSpacingRules` subclass 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 Physical Rules on page 90. 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.

```
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` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying Spacing Rules Tables: tableSpacingRules()

The tableSpacingRules subclass of the Physical Rules class specifies lookup tables for assigning physical spacing rule values.

Table 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. The data for each rule must be enclosed within its own set of parentheses.

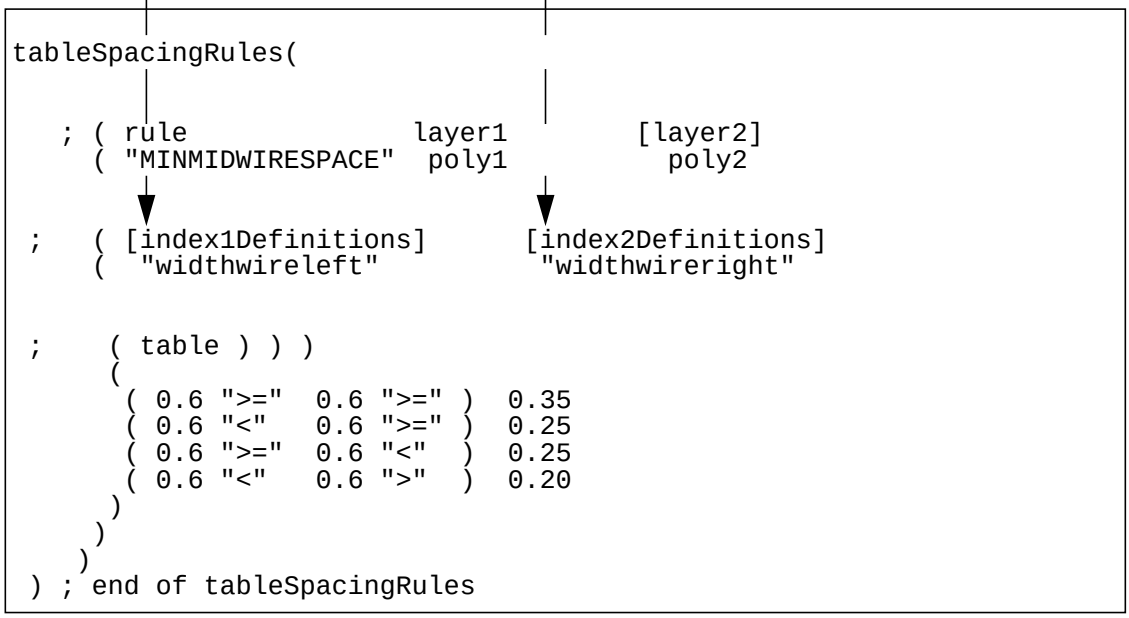
Layer name 1. Specify the layer name, layer number, or layer-purpose pair (enclosed in parentheses) for the first layer on which to apply the rule table. Recommendation: Organize sections according to layer when specifying rules for multiple layers.

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

```
tableSpacingRules(

   ; ( rule              layer1        [layer2]
     ( "MINMIDWIRESPACE"  poly1          poly2


 ;   ( [index1Definitions]      [index2Definitions]
     (  "widthwireleft"          "widthwireright"


 ;    ( table ) ) )
      (
       ( 0.6 ">="  0.6 ">=" )  0.35
       ( 0.6 "<"   0.6 ">=" )  0.25
       ( 0.6 ">="  0.6 "<"  )  0.25
       ( 0.6 "<"   0.6 ">"  )  0.20
      )
     )
    )
 ) ; end of tableSpacingRules
```

Dimension 1 index definitions. Specify the index name, predefined index values, and user-defined match function for the first table dimension for a two-dimensional table or the only dimension for a one-dimensional table. If you do not specify an index name, then the dimension is unnamed. To limit the index values that can be used in the table, specify predefined index values; if you do not specify predefined index values, then any index value can be specified. To employ a match type you define, specify the name of the match function here.

Dimension 2 index definitions. Specify the index definitions for the second dimension in a two-dimensional table.



```
tableSpacingRules(

   ; ( rule              layer1        [layer2]
     ( "MINMIDWIRESPACE"  poly1          poly2

 ;   ( [index1Definitions]      [index2Definitions]
     (  "widthwireleft"          "widthwireright"


 ;    ( table ) ) )
      (
       ( 0.6 ">="  0.6 ">=" )  0.35
       ( 0.6 "<"   0.6 ">=" )  0.25
       ( 0.6 ">="  0.6 "<"  )  0.25
       ( 0.6 "<"   0.6 ">"  )  0.20
      )
     )
    )
 ) ; end of tableSpacingRules
```

Table entries. Specify the table entries as follows:

Index. Specify the index and match type, separated by a space, for the entry. Not specifying the match type selects the default, ==. For a two-dimensional table, specify the second index and match type; separate all parameters by spaces and place the entire index specification in parentheses.

Value. Specify the value to use when a match to the index or indices occurs.

```
tableSpacingRules(

   ; ( rule              layer1        [layer2]
     ( "MINMIDWIRESPACE"  poly1          poly2

 ;   ( [index1Definitions]      [index2Definitions]
     (  "widthwireleft"                "widthwireright"

 ;    ( table ) ) )
      (
       ( 0.6 ">="  0.6 ">=" )  0.35
       ( 0.6 "<"   0.6 ">=" )  0.25
       ( 0.6 ">="  0.6 "<"  )  0.25
       ( 0.6 "<"   0.6 ">"  )  0.20
      )
     )
    )
 ) ; end of tableSpacingRules
```

For more information about the `tableSpacingRules` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying Grid-Snapping Rule: mfgGridResolution()

The `mfgGridResolution` subclass 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.

```
mfgGridResolution(

   ; ( value )

     ( 0.001000 )

  )  ; end of mfgGridResolution
```

For more information about the `mfgGridResolution` subclass, refer to the *Technology File and Display Resource File ASCII Syntax 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 shows rules recognized by Cadence design applications and the applications that use them. You can also specify any rule used by the applications you employ, including any user-defined rules you use in your applications.

Type of Electrical Rule	Rule Name	Applications
The loading capacitance of the layer in picofarads per square micron	`areaCap`	Virtuoso XL, Preview Silicon Ensemble, Preview Gate Ensemble
The amount of current that the default wire width can carry	`currentDensity`	
The capacitance between parallel objects	`edgeCapacitance`	Preview Silicon Ensemble, Preview Gate Ensemble
The fringe capacitance for the layer	`parallelCap`	
The resistivity of the layer in ohms per square	`sheetRes`	Virtuoso XL, Preview Silicon Ensemble, Preview Gate Ensemble

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, along with the electrical characterization rules they define:

```
electricalRules(

orderedCharacterizationRules(
    ; ( rule        layer1  layer2   value )

        ( parallelCap metal1  metal2   2.00 )
        ( parallelCap metal3  metal4   2.00 )
             .
             .
             .
        ( parallelCap prBnd1  prBnd2   1.00 )
    ) ; end of orderedCharacterizationRules

characterizationRules(
    ; ( rule       layer    value )

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

tableCharacterizationRules(
 ; ( rule                        layer1        [layer2]
  ( "ACCURRENTDENSITY PEAK"    "RX"
;   ( [index1Definitions]      [index2Definitions] )
;   ( indexName    ( indexValue ...)   matchType   )
    ( "FREQUENCY"  (1000000.0 1e+08)   nil )
;   ( table ) )
;   ( index       value     index   value ...)
    ( 1000000.0   5e-07     1e+08   4e-07 )
  ) ; end of tableCharacterizationRules

) ; end of electricalRules
```

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

Ordered Characterization Rules subclass. Defines electrical characterization rules in which the order of layers is important. Specifies rule name, layer 1, layer 2, and value. Data for each ordered characterization rule must be enclosed in parentheses. All ordered characterization rules must be enclosed within the parentheses of the subclass enclosure.

Characterization Rules subclass. Defines electrical characterization rules in which the order of layers is not important. Specifies rule name, layer, and value. Data for each characterization rule must be enclosed in parentheses. All characterization rules must be enclosed within the parentheses of the subclass enclosure.

Table Characterization Rules subclass. Defines lookup tables for selecting characterization rules based on conditions. Data for each table characterization rule must be enclosed in parentheses. All table characterization rules must be enclosed within the parentheses of the subclass enclosure.

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

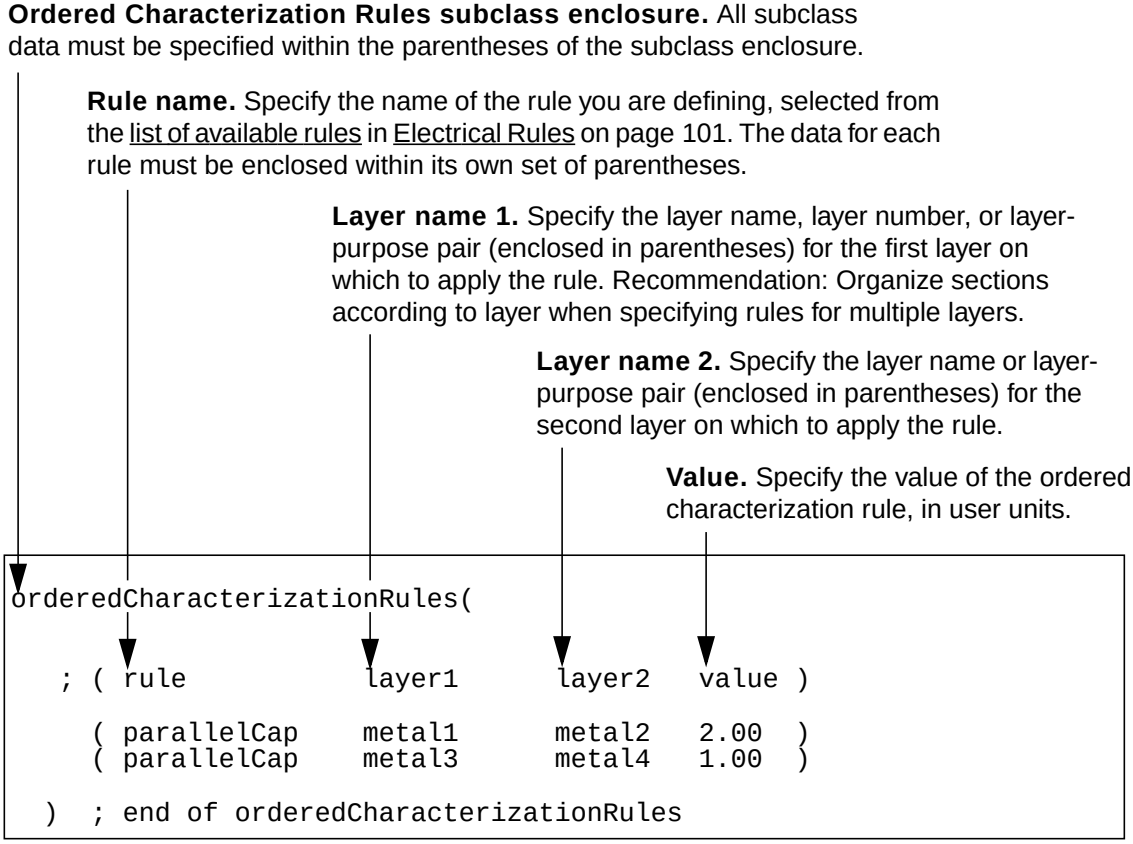
For more information about the `electricalRules` class, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

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.

Specifying Electrical Characterization Rules for Ordered Layers: orderedCharacterizationRules()

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



For more information about the `orderedCharacterizationRules` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying Electrical Characterization Rules: characterizationRules()

The `characterizationRules` subclass 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 Electrical Rules on page 101. 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          metal1      1.4e-4 )
     ( currentDensity   metal1      2.000000 )
     ( edgeCapacitance  metal1      4.0e-11 )
     ( sheetRes         metal1      0.040000 )

  )  ; end of characterizationRules
```

For more information about the `characterizationRules` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying Characterization Rules Tables: tableCharacterizationRules()

The tableCharacterizationRules subclass of the Electrical Rules class specifies lookup tables for assigning electrical characterization rule values.

Table 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. The data for each rule must be enclosed within its own set of parentheses.

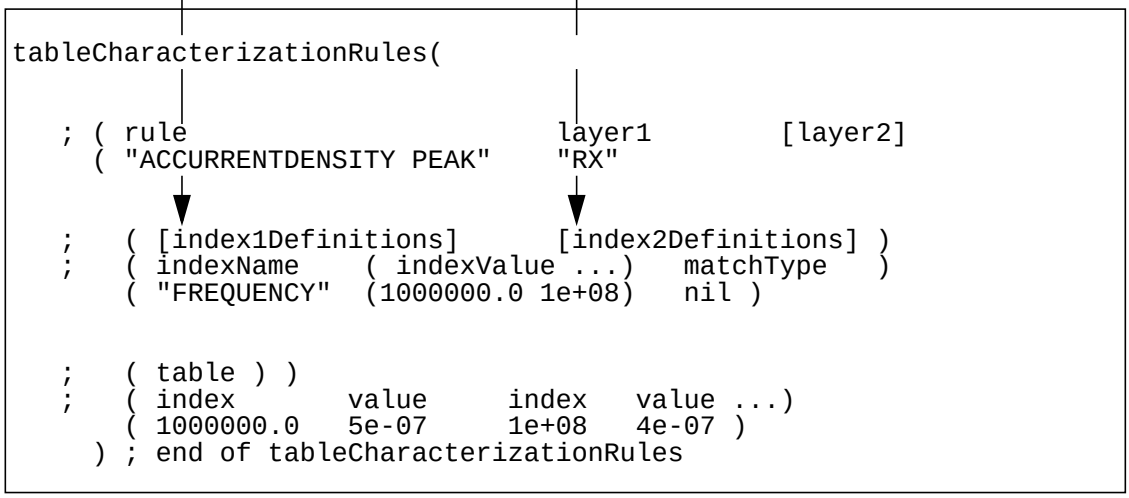
Layer name 1. Specify the layer name, layer number, or layer-purpose pair (enclosed in parentheses) for the first layer on which to apply the rule table. Recommendation: Organize sections according to layer when specifying rules for multiple layers.

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

```
tableCharacterizationRules(

   ; ( rule                        layer1        [layer2]
     ( "ACCURRENTDENSITY PEAK"    "RX"


   ;   ( [index1Definitions]      [index2Definitions] )
   ;   ( indexName    ( indexValue ...)   matchType   )
       ( "FREQUENCY"  (1000000.0 1e+08)   nil )


   ;   ( table ) )
   ;   ( index       value     index   value ...)
       ( 1000000.0   5e-07     1e+08   4e-07 )
     ) ; end of tableCharacterizationRules
```

Dimension 1 index definitions. Specify the index name, predefined index values, and user-defined match function for the first table dimension for a two-dimensional table or the only dimension for a one-dimensional table. If you do not specify an index name, then the dimension is unnamed. To limit the index values that can be used in the table, specify predefined index values; if you do not specify predefined index values, then any index value can be specified. To employ a match type you define, specify the name of the match function here.

Dimension 2 index definitions. Specify the index definitions for the second dimension in a two-dimensional table.

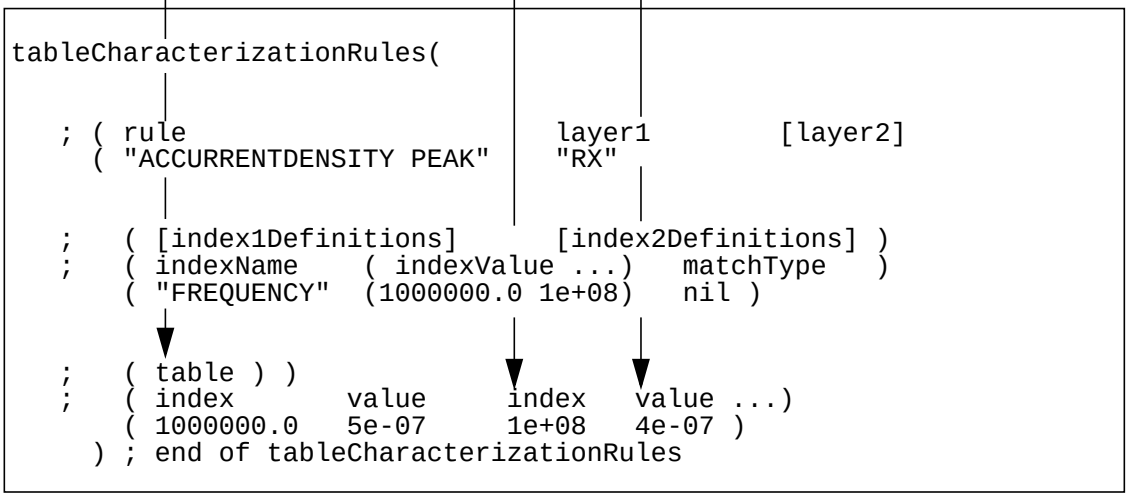


```
tableCharacterizationRules(

   ; ( rule                        layer1        [layer2]
     ( "ACCURRENTDENSITY PEAK"    "RX"

   ;   ( [index1Definitions]      [index2Definitions] )
   ;   ( indexName    ( indexValue ...)   matchType   )
       ( "FREQUENCY"  (1000000.0 1e+08)   nil )


   ;   ( table ) )
   ;   ( index       value     index   value ...)
       ( 1000000.0   5e-07     1e+08   4e-07 )
     ) ; end of tableCharacterizationRules
```

Table entries. Specify the table entries as follows:

Index. Specify the index and match type, separated by a space, for the entry. Not specifying the match type selects the default, ==. For a two-dimensional table, specify the second index and match type; separate all parameters by spaces and place the entire index specification in parentheses.

Value. Specify the value to use when a match to the index or indices occurs.



For more information about the `tableCharacterizationRules` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

5

Creating a Technology File: Specifying 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 110
- Virtuoso XL Layout Editor Rules on page 113
- Virtuoso Compactor Rules on page 117
- Place and Route Rules on page 126

Other technology file classes define the following:

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

Virtuoso Layout Editor Rules

You use the Virtuoso® Layout Editor 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

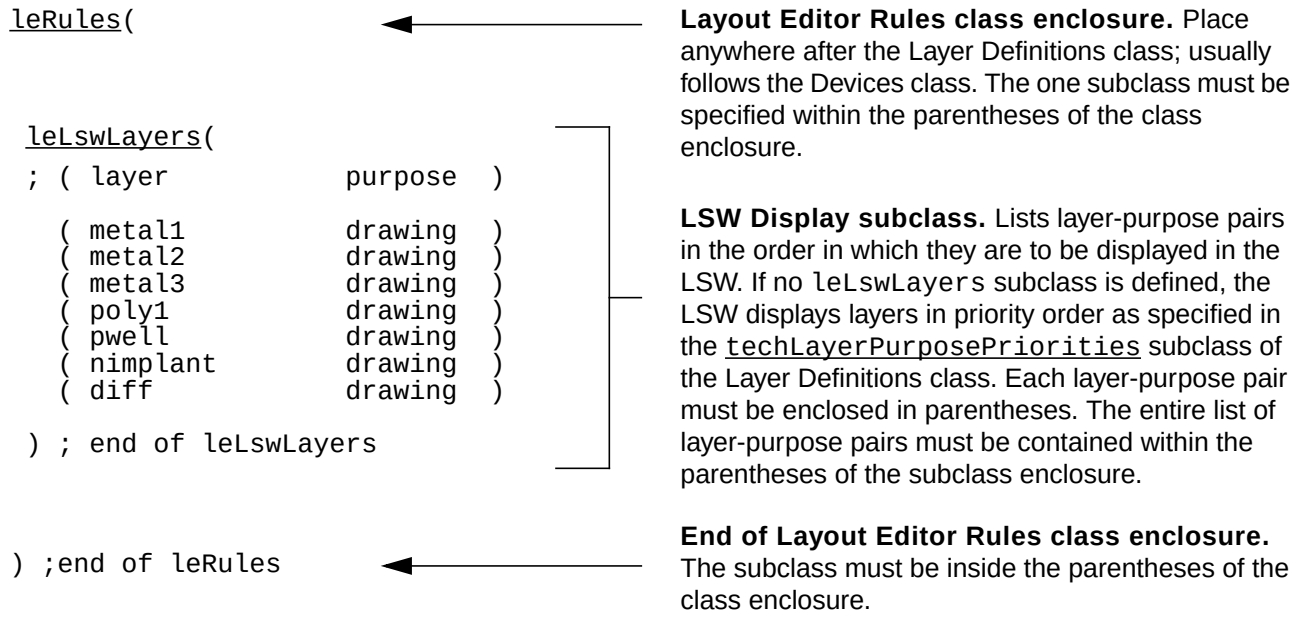
The layout editor displays a tool called the Layer Selection Window (LSW), which is a palette of the layer-purpose pairs you use to draw your designs. You specify layout editor rules in the technology file to establish how layer-purpose pairs are displayed in the LSW. Layout editor rules define the following:

- 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, along with the layout editor rules they define:



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

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.

Specifying LSW Layers Rules: leLswLayers()

The `leLswLayers` subclass 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 `techLayerPurposePriorities` subclass of the Layer Definitions class. See the table below for further information about LSW displays.

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.

```
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`:

leLswLayers Specified in Technology File	Layer-Purpose Pairs Listed in LSW	Layer-Purpose Pairs Listed in LSW's Set Valid Layer Form	Order of Layer-Purpose Pairs Listed in LSW
Yes	All *valid* layers specified in `leLswLayers`	All layers specified in `leLswLayers`	Same as the order specified in `leLswLayers`
No	All *valid* layers listed in the technology library	All layers listed in the technology library	Priority order as specified in the Layer Definitions class, `techLayerPurposePriorities` subclass

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.

For more information about the `leLswLayers` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Virtuoso XL Layout Editor Rules

You use the Virtuoso XL Layout Editor (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 XL Layout Editor User Guide*.

Sample Virtuoso XL Rules Class

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

```
lxRules(

  lxExtractLayers(
  ; layer...

    pwell  ndiff  pdiff  poly1  cont  metal1
    via  metal2  (metal1 drawing1)
    (metal2 drawing1)  via2  metal3

  ) ;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 relative object design (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.

For more information about the `lxRules` class, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying Virtuoso XL Rules

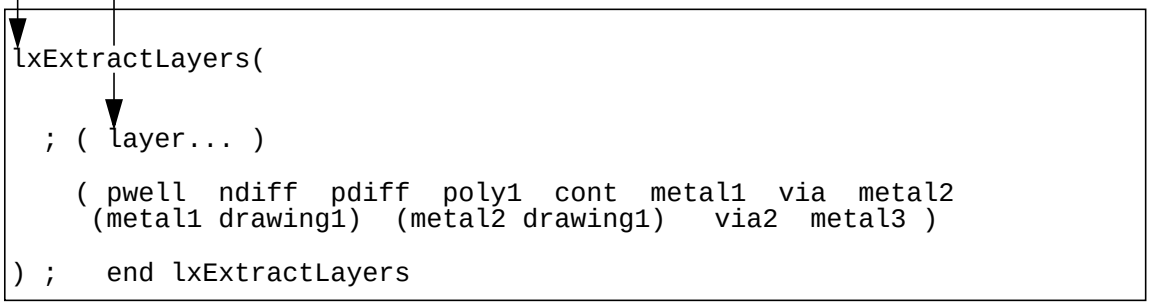
Virtuoso XL rules are required when you are using Virtuoso XL. As illustrated in the sample, the class must be specified by enclosing the subclass definitions within the `lxRules()` class enclosure. The following paragraphs provide detailed information about specifying subclass data.

Specifying Extract Layers Rules: lxExtractLayers()

The `lxExtractLayers` subclass 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.

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.



```
lxExtractLayers(

  ; ( layer... )

    ( pwell  ndiff  pdiff  poly1  cont  metal1  via  metal2
     (metal1 drawing1)  (metal2 drawing1)   via2  metal3 )

) ;   end lxExtractLayers
```

For more information about the `lxExtractLayers` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

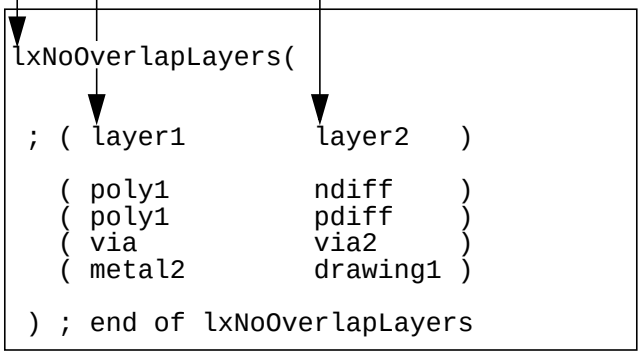
Specifying No Overlap Layers Rules: lxNoOverlapLayers()

The `lxNoOverlapLayers` subclass 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
```

For more information about the `lxNoOverlapLayers` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying Multipart Path Templates: lxMPPTemplates()

The `lxMPPTemplates` subclass 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.

MPP template name. Specify a unique name for the MPP template being defined.

Template definition lists. Specify master path arguments and, optionally, offset subpath arguments, enclosure subpath arguments, and/or subrectangle arguments. These arguments are based on the arguments for the `rodCreatePath` function. For details, see `lxMPPTemplates` in the *Technology File and Display Resource File ASCII Syntax Reference Manual* and the rodCreatePath section of the *Virtuoso Relative Object Design User Guide*.

```
lxMPPTemplates(

 ; ( t_mppTemplateName    l_template)

    ) ;end lxMPPTemplates
```

For more information about the `lxMPPTemplates` subclass, refer to the *Technology File and Display Resource File ASCII Syntax 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, along with the compactor rules they define:

```
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 )]           [WLM] )
    ( "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). 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. 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.

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

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.

Specifying Compactor Layer Usage: compactorLayers()

The `compactorLayers` subclass 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.

Layer name. Specify the layer name or layer-purpose pair (enclosed in parentheses) for the layer for which you are defining the compactor usage. Each layer specified must be defined in the Layer Definitions class.

Usage. Specify the keyword indicating the compactor usage of the layer. The following are the legal keywords:

```
      diffusion   softfence   implant   well
      conduction  hardfence   via       cellBoundary
```

```
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" )

 ) ;end of compactorLayers
```

For more information about the `compactorLayers` subclass, refer to the *Technology File and Display Resource File ASCII Syntax 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.

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 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.

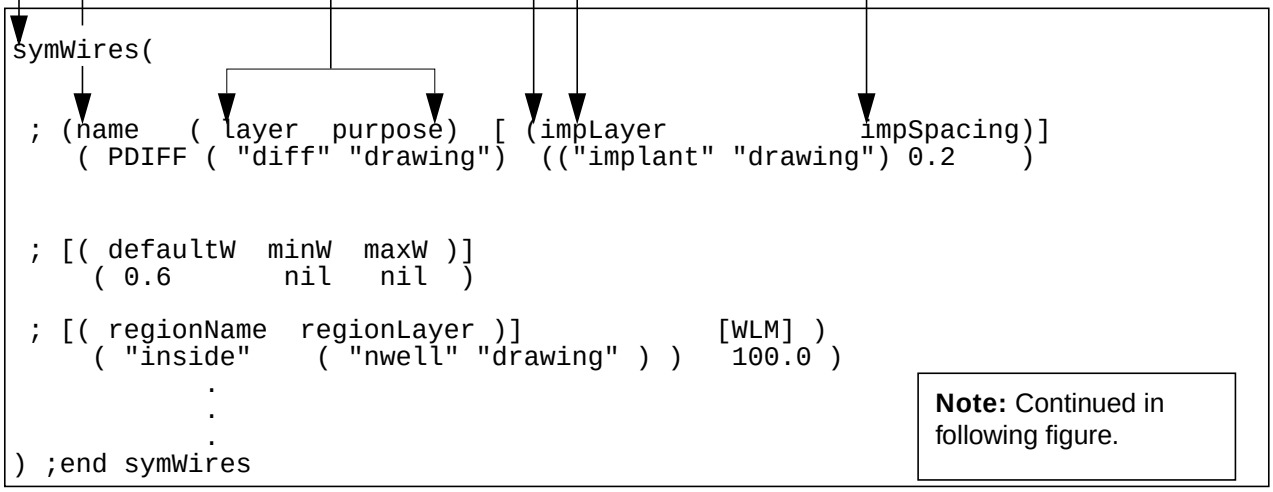
Wire name. Specify the name of the wire to update or create.

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.



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.

```
symWires(

 ; (name   ( layer  purpose)  [ (impLayer            impSpacing)]
    ( PDIFF ( "diff" "drawing")  (("implant" "drawing") 0.2    )

 ; [( defaultW  minW  maxW )]
     ( 0.6       nil   nil  )

 ; [( regionName  regionLayer )]            [WLM] )
     ( "inside"    ( "nwell" "drawing" ) )   100.0 )
             .
             .
             .
) ;end symWires
```

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 )]            [WLM] )
     ( "inside"    ( "nwell" "drawing" ) )   100.0 )
             .
             .
             .
) ;end symWires
```

For more information about the `symWires` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*. For more information about the options you specify, refer to the *Virtuoso Compactor Reference Manual*.

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 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.

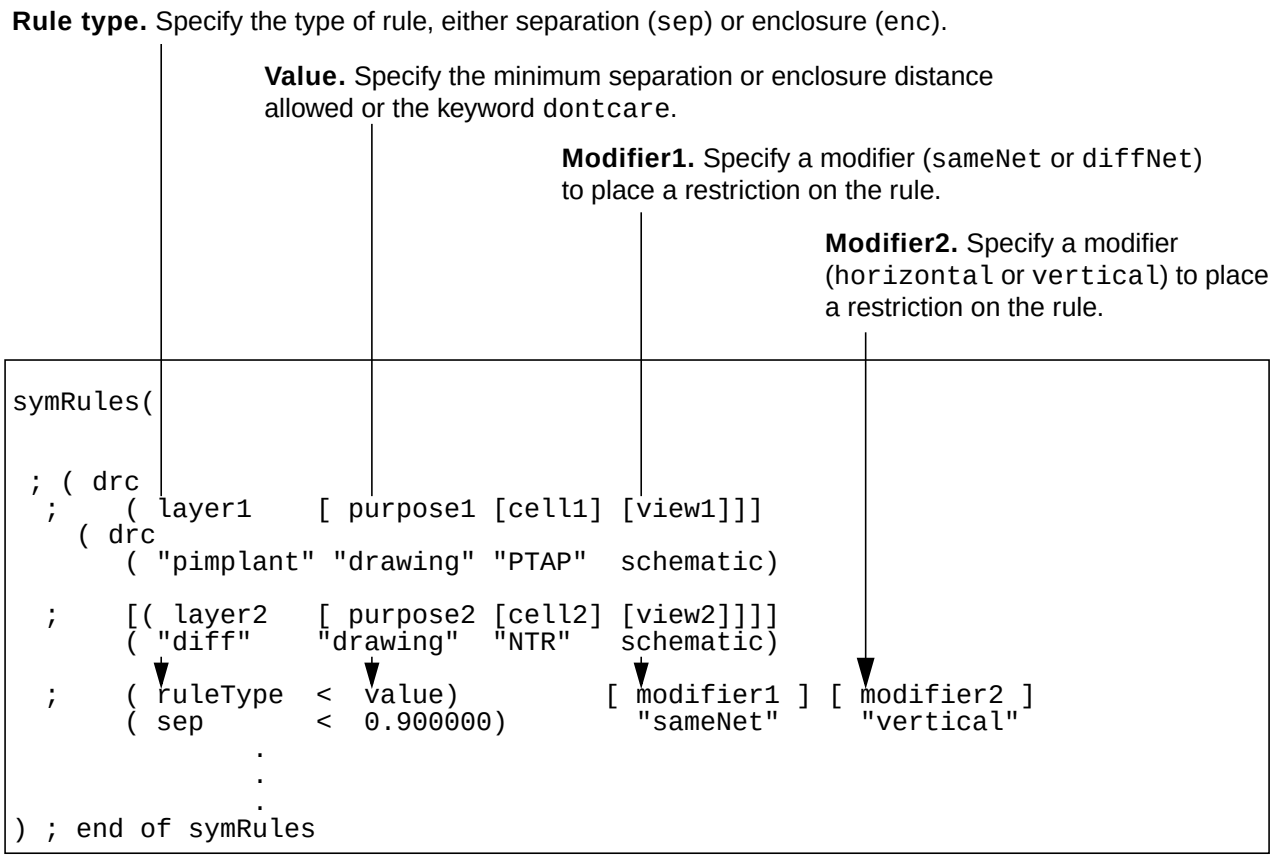
Cell name. Specify the name of the master cell of the component.

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.

```
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.



For more information about the symRules subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Place and Route Rules

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
- Maximum number of stacked vias allowed within a range of layers
- 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

Sample Place and Route Rules Class

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

For more information about the `prRules` class, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

```
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

prMaxStackVias(
; ( value  [ bottomLayer  topLayer ] )
  ( 4        metal1       metal5     )
  ( 2        metal5       metal7     )

) ; end prMaxStackVias
```

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 place-and-route software. The list of layers must be enclosed in parentheses. All stack layer data must be enclosed within the parentheses of the subclass enclosure.

Maximum Stack Vias subclass. Defines the maximum number of stacked vias allowed within a specified layer range. Each maximum number and layer range must be enclosed in parentheses. All maximum stack layer data must be enclosed within the parentheses of the subclass enclosure.

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

    ) ;end prMastersliceLayers
```

Master Slice Layers subclass. Lists master slice layers, from lowest to highest, for use by place-and-route 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.

```
 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_ )

 ;   [( properties ) ... ] )
               .
               .
               .
 ) ; end of prViaRules
```

Via Placement Rules subclass. Defines rules for placing vias with place-and-route 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.

```
prGenViaRules(
 ; ( ruleName    layer
 ; ( lowerPt  upperPt  xPitch  yPitch  resistance )
 ;   layer1   dir1   |  ( g_enc1Overhang1  g_enc1Overhang2 )
 ;   ( wMin wMax overhang metalOverHang )
 ;   layer2   dir2  |   ( g_enc2Overhang1  g_enc2Overhang2 )
   ( wMin wMax overhang metalOverhang )

   ( viagen21   via
   ( -1:1     1:1      1.2     1.2     _NA_       )
     metal1   "horizontal" ( .6   2.0 .6_      .6 )
     metal2   "vertical"   ( .6   2.0 .6       .6 )

 ;   [( properties ) ... ] )
               .
               .
               .
 ) ; end of prGenViaRules
```

Via Generation Rules subclass. Defines rules for generating vias with place-and-route 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.

```
prNonDefaultRules(
;   ( ruleName
;   ( ( layer    width  spacing  [notch] [wExt] [cap] [res] [edgeCap])...)
;   ( viaName   ... )
;   [ ( ( layer1   layer2   minSpace   stack ) ... ) ]
;   [( properties ) ... ] )

    ("NDrule1"
    ( (metal1  3.8    1.5      1.3         )
      ( metal2 4.8    1.6      _NA_        )
      ( metal3 4.8    1.6      _NA_        )     )
    ( ND1M2_M1  ND1M3_M2       )
    ( ( metal1   metal1   1.3        nil   )
      ( via      via2      4.5        nil   )     ) )
) ; end of prNonDefaultRules
```

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.

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

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.

```
prRoutingOffset(
 ; ( layer          offset  )
   ( metal1         0.0     )
   ( metal2         1.2     )
              .
              .
              .
) ; end of prRoutingOffset
```

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.

```
prOverlapLayer(
 ;  layer
    overlap   overlap2


) ; end of prOverlapLayer
```

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 prRules
```

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 Cadence place-and-route tools. 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.

Specifying Routing Layers: prRoutingLayers()

The `prRoutingLayers` subclass 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.

Layer name. Specify the layer name. For each layer you specify, the layer-purpose pair with the purpose `drawing` must be defined in the Layer Definitions class.

Direction. Specify the routing direction from the choices `halfRoute`, `vertical`, or `horizontal`.

```
prRoutingLayers(

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

For more information about the `prRoutingLayers` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying Vias: prViaTypes()

The `prViaTypes` subclass 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
```

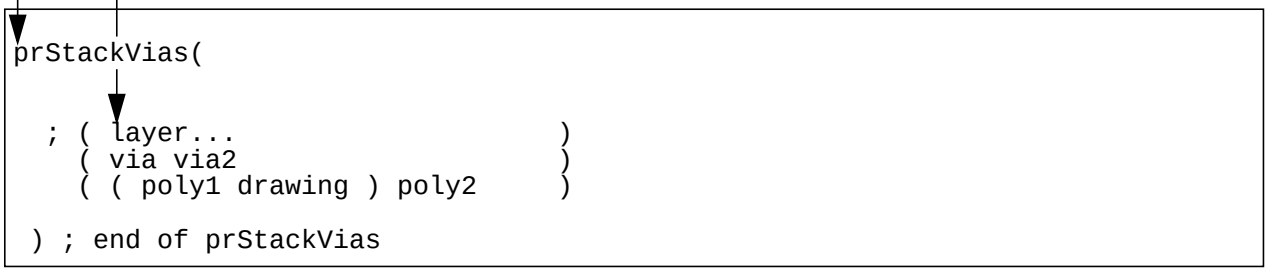
For more information about the `prViaTypes` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying Vias That Can Be Stacked: prStackVias()

The `prStackVias` subclass of the Place and Route Rules class defines the via layers that can be stacked in place-and-route 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.



```
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 ASCII Syntax Reference Manual*.

Specifying The Maximum Number of Stacked Vias within a Layer Range: prMaxStackVias()

The `prMaxStackVias` subclass of the Place and Route Rules class defines the maximum number of stacked vias allowed within a specified layer range.

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

Value. Specify the maximum number of stacked vias.

Bottom layer. Specify the bottom layer of the layer range.

Top layer. Specify the top layer of the layer range.

```
prStackVias(

  ; ( value...[ bottomLayer      topLayer ] )
    ( 4         metal1           metal5     )
    ( 2         metal5           metal7     )

 ) ; end of prMaxStackVias
```

For more information about the `prMaxStackVias` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Specifying Master Slice Layers: prMastersliceLayers()

The `prMastersliceLayers` subclass of the Place and Route Rules class defines the master slice layers for use in place-and-route 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.

```
prMastersliceLayers(

  ; ( layer...                    )
    ( diff  poly1                 )
                .
                .
                .

 ) ; end of prMastersliceLayers
```

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

Specifying Via Rules: prViaRules()

The `prViaRules` subclass of the Place and Route Rules class defines the rules for placing vias using place-and-route software.

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.

```
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_ )


 ;   [( properties ) ... ] )
               .
               .
               .
 ) ; end of prViaRules
```

Note: Continued in following figures.

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. To define a turn via, specify the same layer for *`layer1`* and *`layer2`*.

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

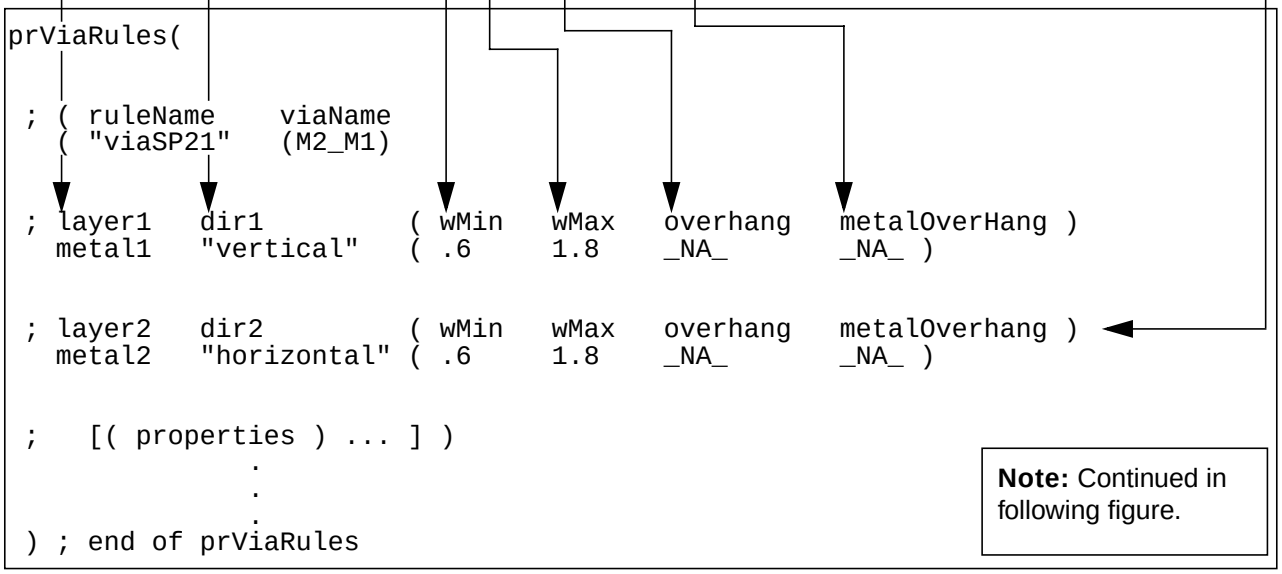
Minimum width. Specify the minimum allowable wire width, in user units.

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.

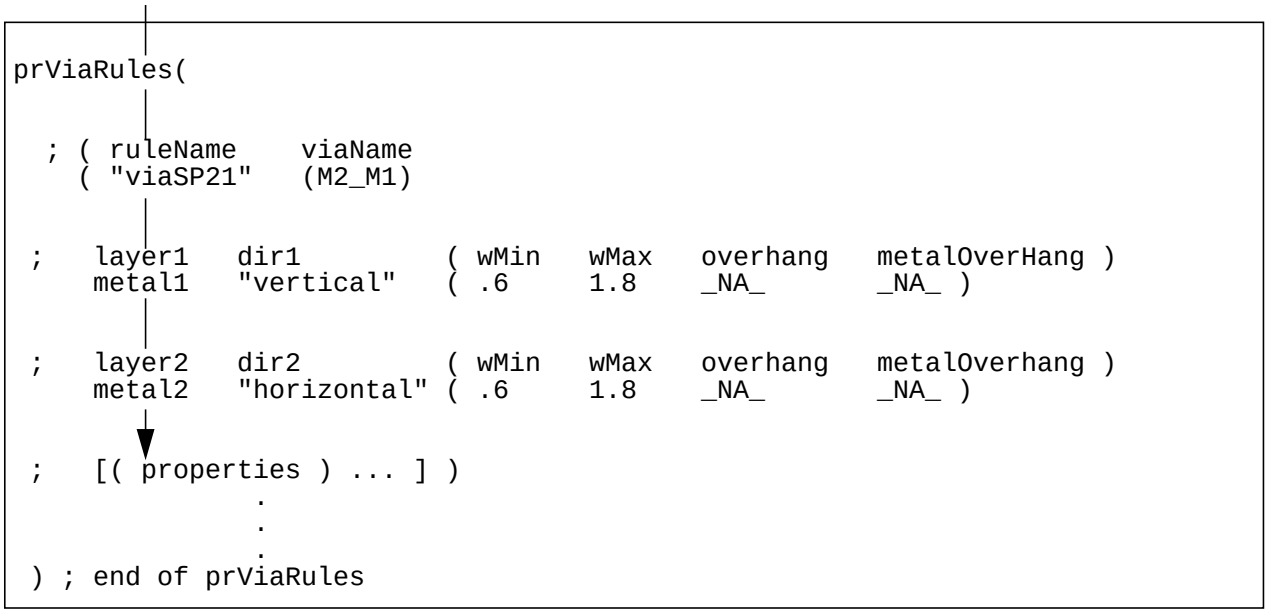
Layer 2 specifications. Specify the same set of parameters for the layer for the bottom of the via as for the layer for the top of the via.



```
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_ )

 ;   [( properties ) ... ] )
               .
               .
               .
 ) ; end of prViaRules
```

Note: Continued in following figure.

Properties. Specify properties for the via, each property with its value, enclosed in parentheses and separated by a space.



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

```
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.    _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 ASCII Syntax 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 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.

```
prGenViaRules(

  ; ( ruleName    layer
    ( viagen21   via


 ; ( lowerPt  upperPt  xPitch  yPitch  resistance )
   ( -1:1     1:1      1.2     1.2     _NA_       )


 ; layer1   dir1         |  ( g_encl1Overhang1  g_enc1Overhang2 )
   metal1   "horizontal"


 ; ( wMin   wMax   overhang   metalOverHang )
   ( .6     2.0    .6_        .6             )


 ; layer2   dir2         |  ( g_encl2Overhang1  g_enc2Overhang2 )
   metal2   "vertical"


 ; ( wMin   wMax   overhang   metalOverhang )
   ( .6     2.0    .6         .6             )


 ;   [( properties ) ... ] )
             .
             .
             .
    ) ;end prGenViaRules
```

Note: Continued in following figures.

Lower point. Specify the lower x,y coordinate of the bounding box for the via layer.

Upper point. Specify the upper x,y coordinate of the bounding box for 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

 ; ( lowerPt  upperPt  xPitch  yPitch  resistance )
   ( -1:1     1:1      1.2     1.2     _NA_       )


 ; layer1   dir1         |  ( g_encl1Overhang1  g_enc1Overhang2 )
   metal1   "horizontal"


 ; ( wMin   wMax   overhang   metalOverHang )
   ( .6     2.0    .6_        .6            )


 ; layer2   dir2         |  ( g_encl2Overhang1  g_enc2Overhang2 )
   metal2   "vertical"


 ; ( wMin   wMax   overhang   metalOverhang )
   ( .6     2.0    .6         .6            )


 ;   [( properties ) ... ] )
             .
             .
             .
    ) ;end prGenViaRules
```

Note: Continued in following figures.

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`). **Note:** Specify a direction *or* enclosure.

Enclosure overhang 1. Specify the enclosure overhang for two opposite sides of the via. **Note:** Specify a direction *or* enclosure.

Enclosure overhang 2. Specify the enclosure overhang for the other two opposite sides of the via. **Note:** Specify a direction *or* enclosure.

```
prGenViaRules(

  ; ( ruleName    layer
    ( viagen21    via

 ; ( lowerPt  upperPt  xPitch  yPitch  resistance )
   ( -1:1     1:1      1.2     1.2     _NA_       )

 ; layer1   dir1          |  ( g_encl1Overhang1  g_enc1Overhang2 )
   metal1   "horizontal"


 ; ( wMin   wMax   overhang   metalOverHang )
   ( .6     2.0    .6_        .6             )


 ; layer2   dir2          |  ( g_encl2Overhang1  g_enc2Overhang2 )
   metal2   "vertical"


 ; ( wMin   wMax   overhang   metalOverhang )
   ( .6     2.0    .6         .6             )


 ;   [( properties ) ... ] )
               .
               .
               .
 ) ; end of prGenViaRules
```

Note: Continued in following figures.

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. **Note:** Specify only when you specify a direction.

Metal overhang. Specify the minimum overhang of the via to the wire. **Note:** Specify only when you specify a direction.

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

```
prGenViaRules(

 ; ( ruleName    layer
   ( viagen21    via

; ( lowerPt  upperPt  xPitch  yPitch  resistance )
  ( -1:1     1:1      1.2     1.2     _NA_       )

; layer1   dir1          |  ( g_encl1Overhang1  g_enc1Overhang2 )
  metal1   "horizontal"

; ( wMin   wMax   overhang   metalOverHang )
  ( .6     2.0    .6_        .6             )

; layer2   dir2          |  ( g_encl2Overhang1  g_enc2Overhang2 )
  metal2   "vertical"

; ( wMin   wMax   overhang   metalOverhang )
  ( .6     2.0    .6         .6             )

;   [( properties ) ... ] )
            .
            .
            .
   ) ;end prGenViaRules
```

Note: Continued in following figure.

Properties. Specify properties for the via, each property with its value, enclosed in parentheses and separated by a space.

```
prGenViaRules(

  ; ( ruleName    layer
    ( viagen21    via

 ; ( lowerPt  upperPt  xPitch  yPitch  resistance )
   ( -1:1     1:1      1.2     1.2     _NA_       )

 ; layer1   dir1         |  ( g_encl1Overhang1  g_enc1Overhang2 )
   metal1   "horizontal"

 ; ( wMin   wMax   overhang   metalOverHang )
   ( .6     2.0    .6_        .6            )

 ; layer2   dir2         |  ( g_encl2Overhang1  g_enc2Overhang2 )
   metal2   "vertical"

 ; ( wMin   wMax   overhang   metalOverhang )
   ( .6     2.0    .6         .6            )

 ;   [( properties ) ... ] )
             .
             .
             .
    ) ;end prGenViaRules
```

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

```
prViaRules(
...
)
prGenViaRules(
 ; ( viaRuleName
 ; viaLayer  ( lowerPt   upperPt   xPitch   yPitch   resistance        )
 ; layer1   dir1          ( wMin   wMax   overhang   metalOverhang  )
 ; layer2   dir2          ( wMin   wMax   overhang   metalOverhang  )

   ( viagen21
   via       ( -1:1      1:1       1.2      1.2      _NA_              )
   metal1   "horizontal"  ( .6     20.0   .6         .6              )
   metal2   "vertical"     ( .6     20.0   .6         .6              ) )
 )
```

This code specifies that the router generate a via with the following dimensions:

- The lower point of a via cut is `-1,1`; the upper point is `1,1`

- 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 ASCII Syntax Reference Manual*.

Specifying Nondefault Rules: prNonDefaultRules()

Nondefault rules control place-and-route 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 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.

```
prNonDefaultRules(

  ; ( ruleName
    ( "NDrule1"

 ; ( ( layer    width  spacing  [notch]] [wExt] [cap] [res] [edgeCap]   )... )
   ( ( metal1   3.8    1.5      1.3                                     )
     ( metal2   4.8    1.6                                              )
     ( metal3   4.8    1.6                                              )    )

 ;   ( viaName   ... )
     ( ND1M2_M1  ND1M3_M2  )

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

 ;   [( properties ) ... ] )
               .
               .
               .
 ) ; 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.

Wire extension. Specify the wire extension allowed for the layer, in user units.

Capacitance. Specify the capacitance allowed for the layer, in user units.

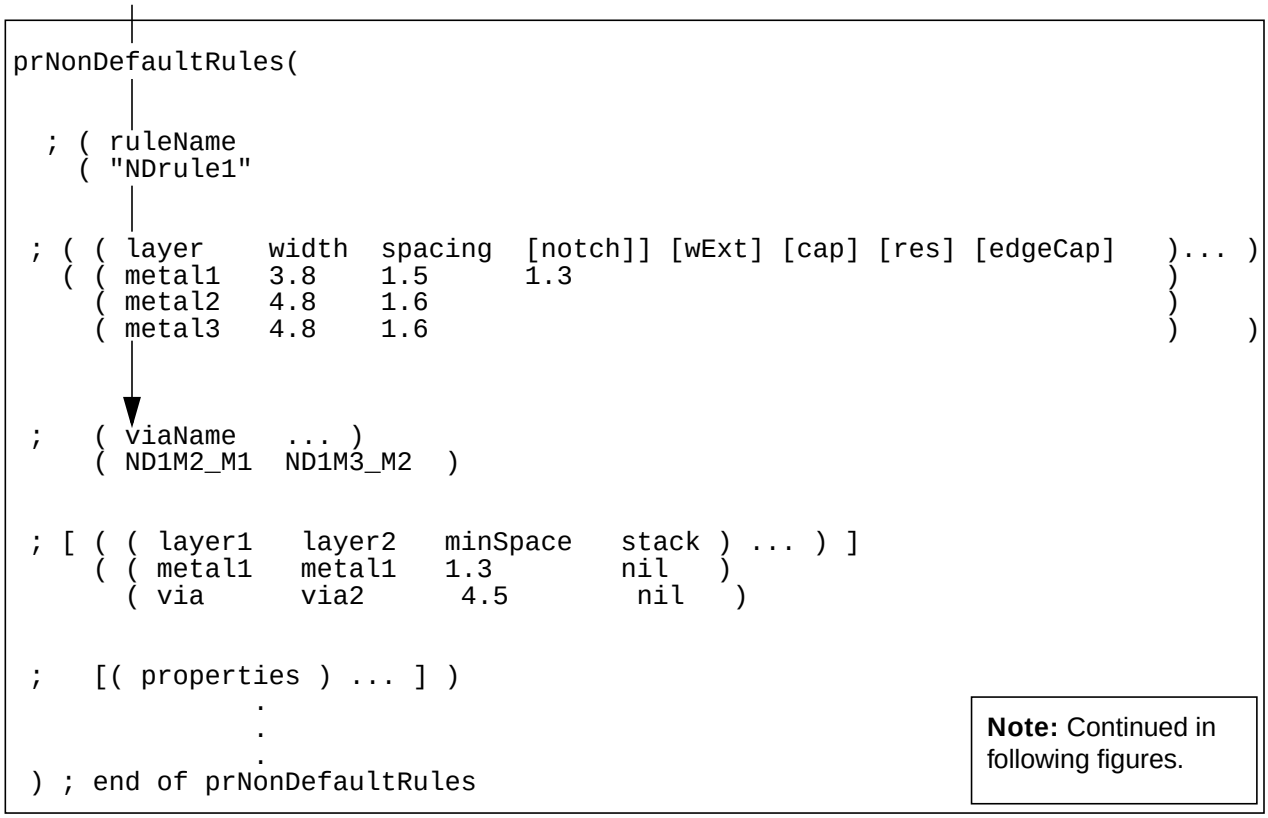
Resistance. Specify the resistance allowed for the layer, in user units.

Edge capacitance. Specify the edge capacitance allowed for the layer, in user units.

```
prNonDefaultRules(

  ; ( ruleName
    ( "NDrule1"

  ; ( ( layer    width  spacing  [notch]] [wExt] [cap] [res] [edgeCap]   )... )
    ( ( metal1   3.8    1.5      1.3                                     )
      ( metal2   4.8    1.6                                              )
      ( metal3   4.8    1.6                                              )    )


  ;   ( viaName   ... )
      ( ND1M2_M1  ND1M3_M2  )


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


  ;   [( properties ) ... ] )
                .
                .
                .
  ) ; end of prNonDefaultRules
```

Note: Continued in following figures.

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.



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.

```
prNonDefaultRules(

  ; ( ruleName
    ( "NDrule1"

 ; ( ( layer    width  spacing  [notch]] [wExt] [cap] [res] [edgeCap]   )... )
   ( ( metal1   3.8    1.5      1.3                                     )
     ( metal2   4.8    1.6                                              )
     ( metal3   4.8    1.6                                              )    )

 ;   ( viaName   ... )
     ( ND1M2_M1  ND1M3_M2  )

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


 ;   [( properties ) ... ] )
               .
               .
               .
 ) ; end of prNonDefaultRules
```

Note: Continued in following figure.

Properties. Specify properties for the via, each property with its value, enclosed in parentheses and separated by a space.

```
prNonDefaultRules(

  ; ( ruleName
    ( "NDrule1"

 ; ( ( layer    width  spacing  [notch]] [wExt] [cap] [res] [edgeCap]   )... )
   ( ( metal1   3.8    1.5      1.3                                     )
     ( metal2   4.8    1.6                                              )
     ( metal3   4.8    1.6                                              )    )

 ;   ( viaName   ... )
     ( ND1M2_M1  ND1M3_M2  )

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

 ;   [( properties ) ... ] )
               .
               .
               .
 ) ; end of prNonDefaultRules
```

For more information about the `prNonDefaultRules` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Setting Routing Pitch: prRoutingPitch()

Routing pitch is the minimum allowable spacing, center-to-center, between two geometries on different nets.

The `prRoutingPitch` subclass 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.

```
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 ASCII Syntax Reference Manual*.

Setting Routing Offset: prRoutingOffset()

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 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 ASCII Syntax Reference Manual*.

Specifying Overlap Layers: prOverlapLayer()

Overlap layers indicate where cells cannot overlap.

The `prOverlapLayer` subclass 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.

```
prOverlapLayer(

  ; ( layer...          )
    ( overlap   overlap2 ).

    ) ;end prOverlapLayer
```

For more information about the `prOverlapLayer` subclass, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

6

Creating a Display Resource File

This chapter discusses the following:

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 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 ASCII Syntax Reference Manual*.

```
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.

```
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
```

Define Display Packet section. Defines the display packets used with various display devices.

```
drDefinePacketAlias(
;( displayName  packetAlias        packetName    )
 ( display      blsol1             blacksolid_S  )
 ( display      blue2              blue          )
 ( display      b3                 blue          )
               .
               .
               .
 ) ; end of drDefinePacketAlias
```

Define Display Packet Alias section. Defines alias names for the specified display packet names.

Specifying Display Resources

This section provides detailed information about specifying display resources in a display resource file.

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.

```
drDefineDisplay(

 ; ( displayName )
   ( display1    )
   ( display2    )
   ( plotter1    )
   ( plotter2    )
         .
         .
         .
) ;  end drDefineDisplay
```

For more information about the `drDefineDisplay` section, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Commonly Used Display Devices

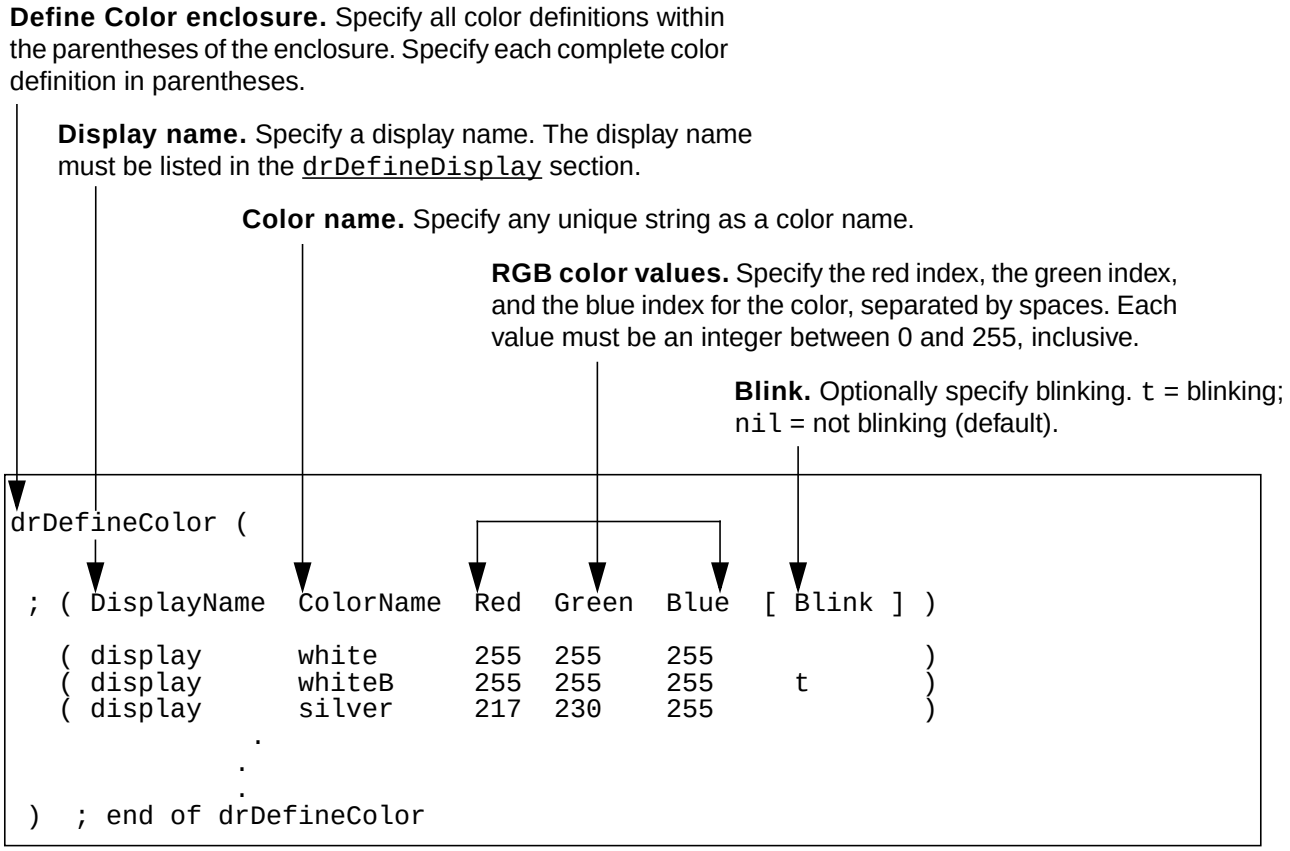
The following table lists commonly used display devices.

Device Name	Type
display	Color monitor
hp6	Hewlett-Packard 6-carousel pen plotters
hp8	Hewlett-Packard 8-carousel pen plotters
psb	PostScript black-and-white plotters
versatecb	Versatec and CalComp black-and-white plotters
versatecc	Versatec and CalComp color plotters
XBlackWhite	Black-and-white X Window System monitors
X4PlaneColor	4-plane color X Window System monitors

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.



For more information about the `drDefineColor` section, refer to the *Technology File and Display Resource File ASCII Syntax 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)
                               (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 the `drDefineStipple` section, refer to the *Technology File and Display Resource File ASCII Syntax 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.

```
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) )
 ( display       dashDot       1       (1 1 1 0 0 1 0 0) )
 ( display       shortDash     1       (1 1 0 0) )
 ( display       doubleDash    1       (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) )
                .
                .
                .
 )  ; end of drDefineLineStyle)
```

For more information about the `drDefineLineStyle` section, refer to the *Technology File and Display Resource File ASCII Syntax 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 162.

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 164. 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)
```

For more information about the `drDefinePacket` section, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Packet Naming Conventions

Cadence recommends 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. The packet name must not contain spaces, and the underscore character (_) must precede the SLNB extension.

color[*stipple*][*line*][_[S][L][N][B]]

where:

color — specifies the fill color and/or the outline color in the following format:

[*fill*] [*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 `drDefineColor`), specify the one color when the outline color name is the same as the fill color name with a B appended (for example, `white` and `whiteB`). If the outline color name is constructed in any other way (for example, `white2`), specify both colors (`whitewhite2`).

stipple — specifies a stipple pattern name. This field is optional; no entry indicates the default stipple pattern name, `blank`.

line — specifies a line style name. This field is optional; no entry indicates the default line style name, `solid`. No entry and the `N` extension to the packet name indicate that the line style name is `none`.

_SLNB extension

`S` indicates that the stipple pattern name is specified and the line style name is the default (`solid`).

`SN` indicates that the stipple pattern name is specified and the line style name is `none`.

`L` indicates that the line style name is specified and the stipple pattern name is the default (`blank`).

`N` indicates that the line style name is `none`.

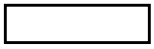
`B` indicates that the outline color is blinking. Only the outline color can be a blinking color; the fill color must be non-blinking. (Colors are defined with `drDefineColor`; for complete syntax information, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.)

The following table shows sample combinations of color, stipple, and line resources and the packet names correctly built according to the packet naming conventions.

Fill Color	Outline Color	Stipple Pattern	Line Style	Packet Name
blue	blue	blank	solid	`blue`
blue	blue	blank	dashed	`bluedashed_L`
blue	blue	solid	solid	`bluesolid_S`
blue	blue	metal1S	solid	`bluemetal1S_S`
blue	blue	metal1S	none	`bluemetal1S_SN`
cream	white	contp	solid	`creamwhitecontp_S`
green	green	brick	mLine	`greenbrickmLine`
red	red	X	thickLine	`redXthickLine`
red	blinking red named redB	x	solid	`redx_SB`
red	blinking red named red2	solid	none	`redred2solid_SNB`

Fill Styles

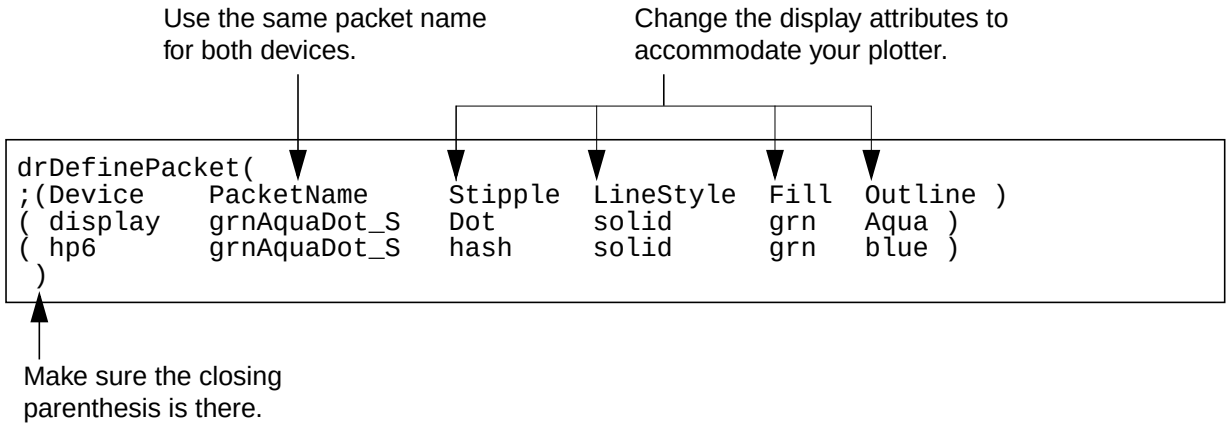
The following table defines the fill styles.

Fill Style	Result
`outline`	
`solid`	
`x`	
`stipple`	 (The stipple pattern specified determines the fill.)
`outline stipple`	 (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.

Display name. Specify a display name. The display name must be listed in the `drDefineDisplay` section.

Alias name. Specify any unique string as an alias for the display packet name.

Display packet name. 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 `drDefinePacket` section.

```
drDefinePacketAlias

 ( displayName  packetAlias        packetName    )
 ( display      blsol1             blacksolid_S  )
 ( display      blue2              blue          )
 ( display      b3                 blue          )
    .
    .
    .
 ) ; end of drDefinePacketAlias
```

For more information about the `drDefinePacketAlias` section, refer to the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

7

Preparing Files for Use with a Design

This chapter discusses the following:

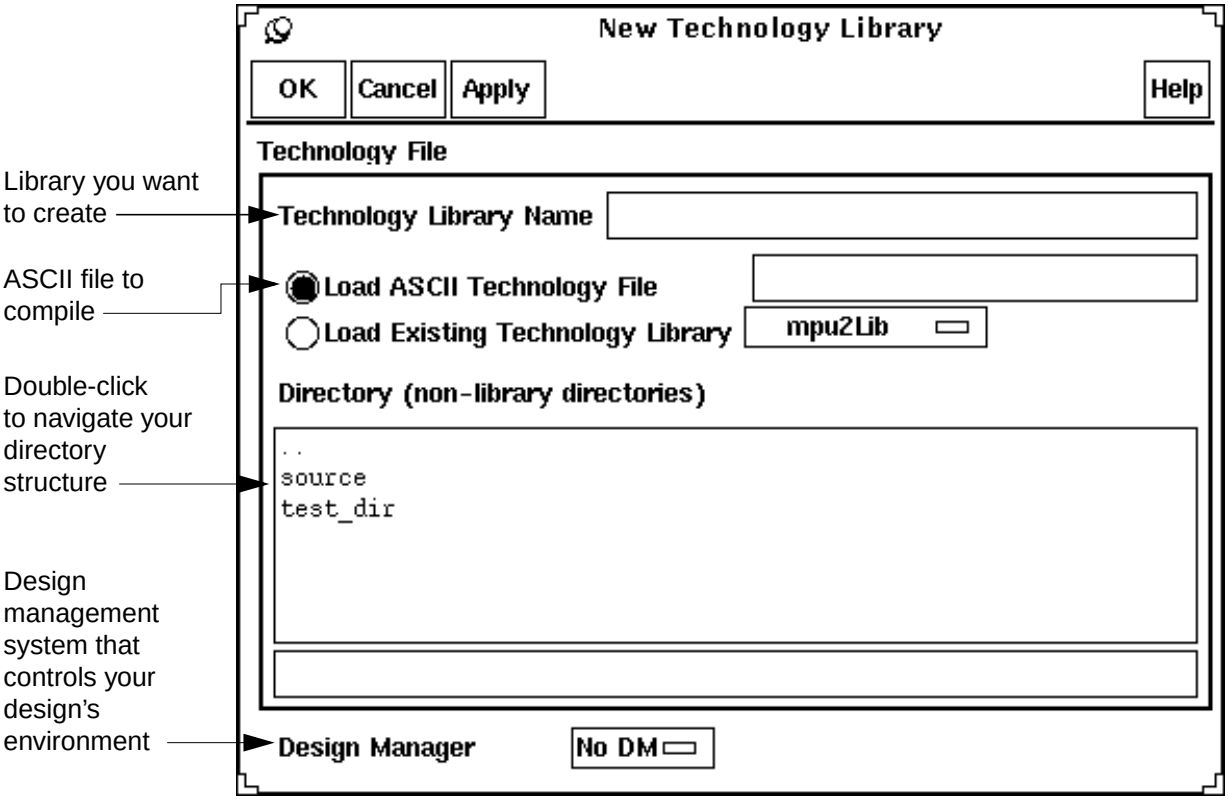
Generating a Technology Library

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 Technology File Tool Box, choose *New*.

   The New Technology Library form appears.



   For a description of this form, see Appendix A.

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 technology library.

- 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 DFII CIW and the `techManager.log` file tell you that check-in of the technology library files and cellviews is skipped because of preferences.

   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*.

   For more information about setting check-in options with DFII, see "Setting Automatic Checkout and Checkin Preferences" in the *Design Framework II User Guide*.

7. Click *OK*.

   The system compiles the ASCII file and creates the new 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 technology 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.

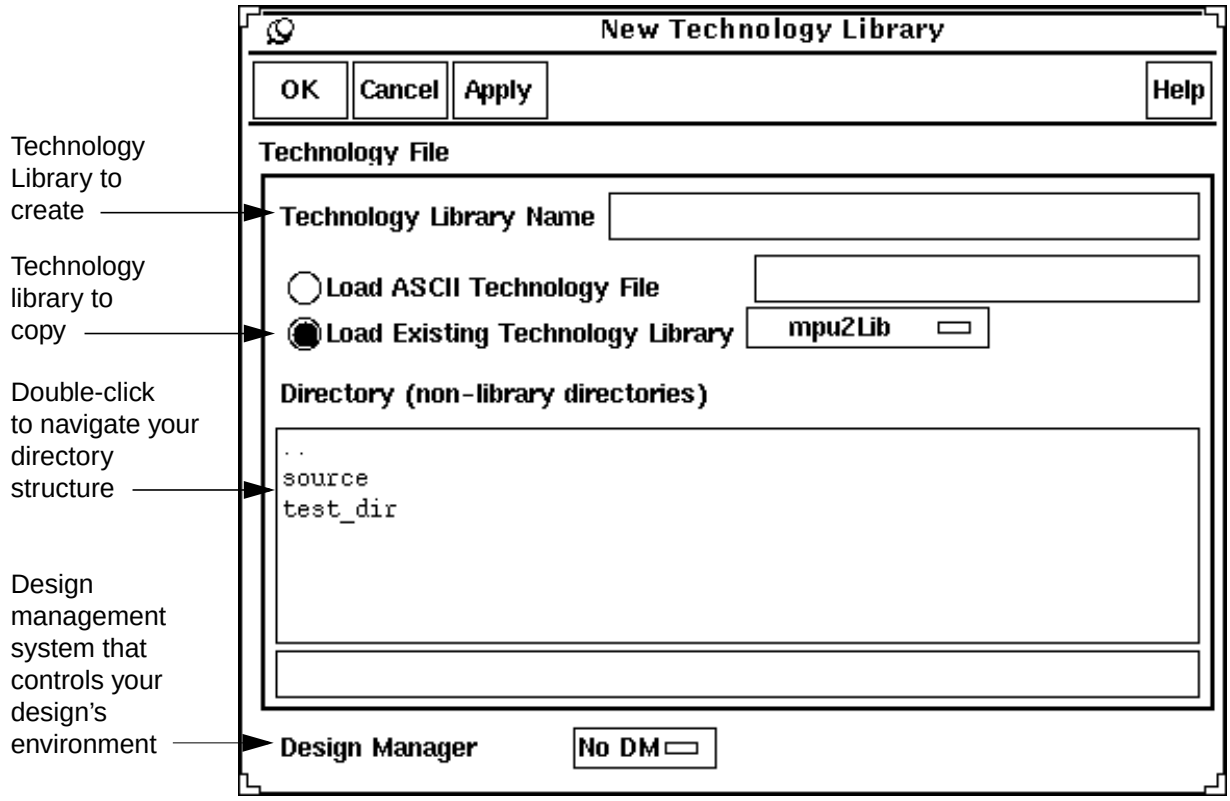
To use the new technology library, you must attach it to a Cadence design or design library. For more information, refer to "Attaching a Technology Library to a Design" on page 174.

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 Technology File Tool Box, choose *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 technology 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 DFII CIW and the `techManager.log` file tell you that check-in of the technology library files and cellviews is skipped because of preferences.

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*.

For more information about setting check-in options with DFII, see "Setting Automatic Checkout and Checkin Preferences" in the *Design Framework II User Guide*.

7. Click *OK*.

   The system copies the binary technology file `techfile.cds` and the devices from the existing library to the new technology library.

   Note: The name of the binary file must be `techfile.cds`. Do not change the name of the file. A technology library can contain only one binary technology file.

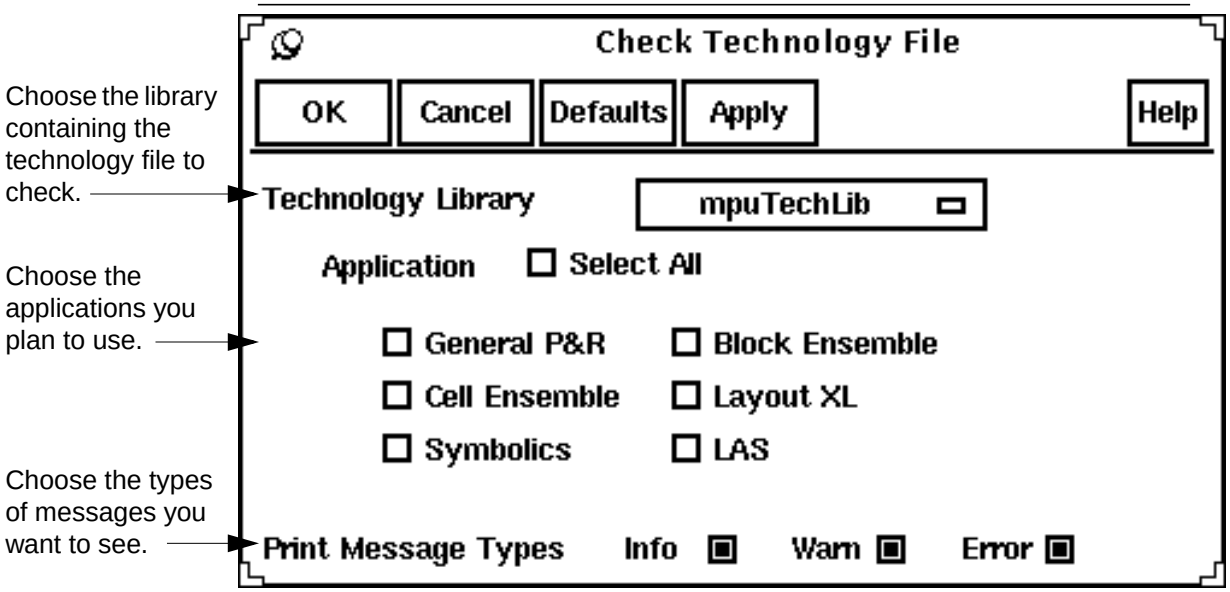
To use the new technology library, you must attach it to a Cadence design or design library. Refer to "Attaching a Technology Library to a Design" on page 174 for more information.

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 Cadence applications.

1. From the Technology File Tool Box, choose *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* buttons, choose the applications you plan to use.
4. From the *Print Message Types* 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. It then logs the results in the `techManager.log` file in your home directory and also displays the results in a window that allows you to save the report, search the report, open another file, and turn auto update on and off. An example is shown below.

   Correct the technology data to conform to the requirements of the applications, regenerate the technology library, and recheck your data for conformance to application requirements.

]

/tmp/CheckTeca27098

File Help

```
\o Check Technology File
\o Library Name: LE
\o Message severity: ("error" "warn" "info")
\o Application: Symbolics
\o
\o
\o ************************************************************
\o *    "Compactor" Technology File Checker                     *
\o ************************************************************
\o
\o
\o
\o Technology Library Name      = LE
\o Technology File Name = techfile.cds
\o
\o
\o
\o Database units per user units:
\o       DBUPerUU = 1000
\o
\o Manufacturing Grid Resolution:
\o       mfgGridResolution = 0.100 user units
\o       MGRInDBU = 100 database unit
\o
\o
\o Checking Rules/Layers/Wires/Devices...
\w *WARNING* lasWires (#2): `("pdiff" "pdiffWire")'
\w       LAS wire should be defined in symWire section.
\o
\o ************************************************************
\o *   Summary Report For Compactor Technology File Checker     *
\o ************************************************************
\o       0 ERROR messages.
\o       1 WARN messages.
\o       0 INFO messages.
\o
```

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.

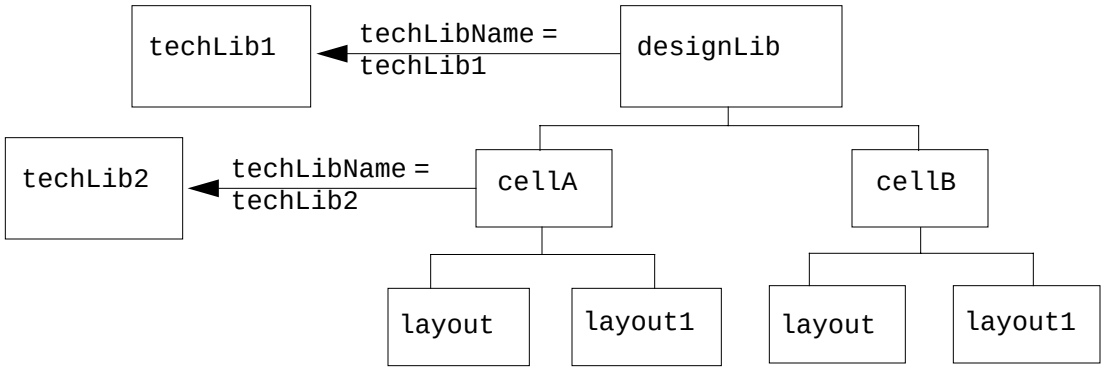
  The `techLibName` property identifies the technology library attached to the library, cell, or cellview.

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`.

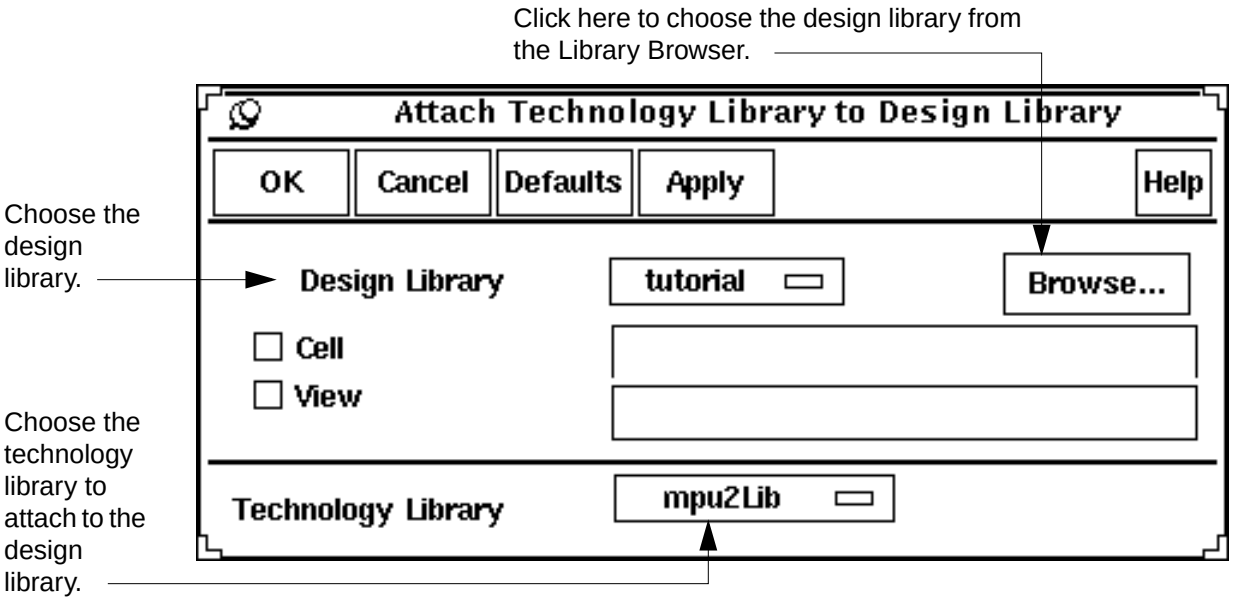


Attaching a Technology Library to a Design Library

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 Technology File Tool Box, choose *Attach*.

   The Attach Technology Library to Design Library 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 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 with DFII, see "Setting Automatic Checkout and Checkin Preferences" in the *Design Framework II User Guide*.

   Upon completion of the attach command, a message similar to the following appears in the DFII CIW and in your `techManager.log` file:

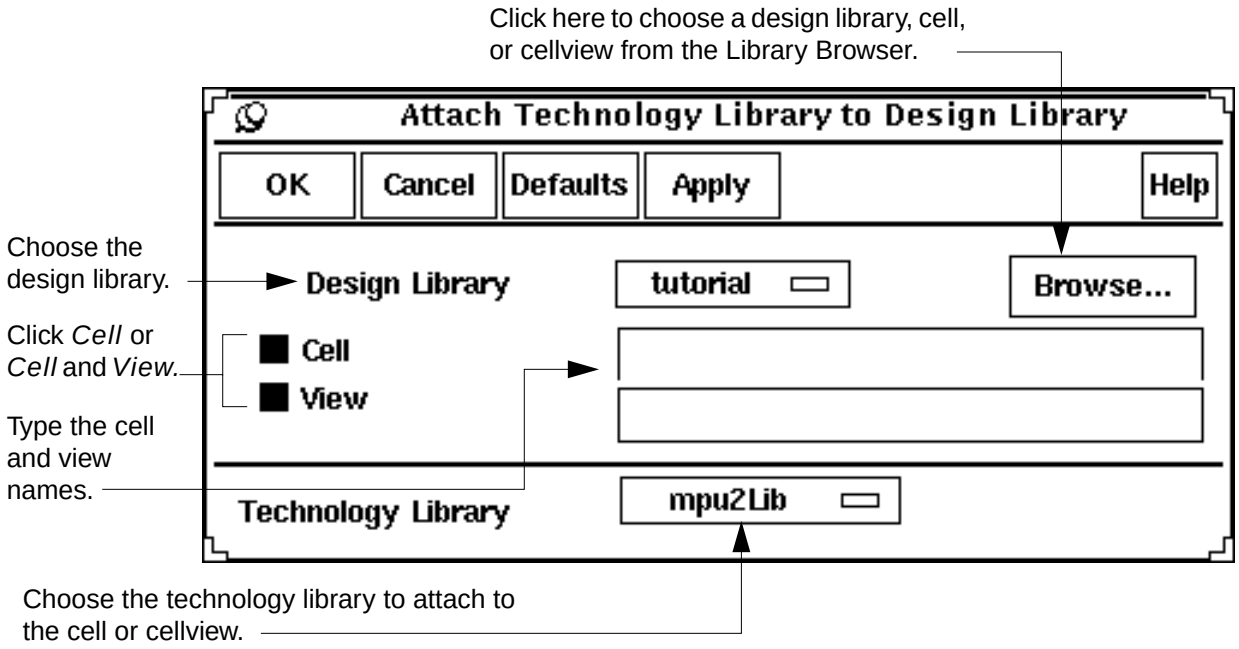
```
Processing (designLib CellA layout) for device update.
Changed device reference (techLib3 M1_P symbolic) to
     (cellTechLib M1_P symbolic).
Processing (designLib CellB layout) for device update.
Changed device reference (techLib3 M2_M1 symbolic) to
     (cellTechLib M2_M1 symbolic).
Design library '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 Technology File Tool Box, choose *Attach*.

The Attach Technology Library to Design Library form appears.



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 with DFII, see "Setting Automatic Checkout and Checkin Preferences" in the *Design Framework II User Guide*.

Upon completion of the attach command, a message similar to the following appears in the DFII CIW and in your `techManager.log` file:

```
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`

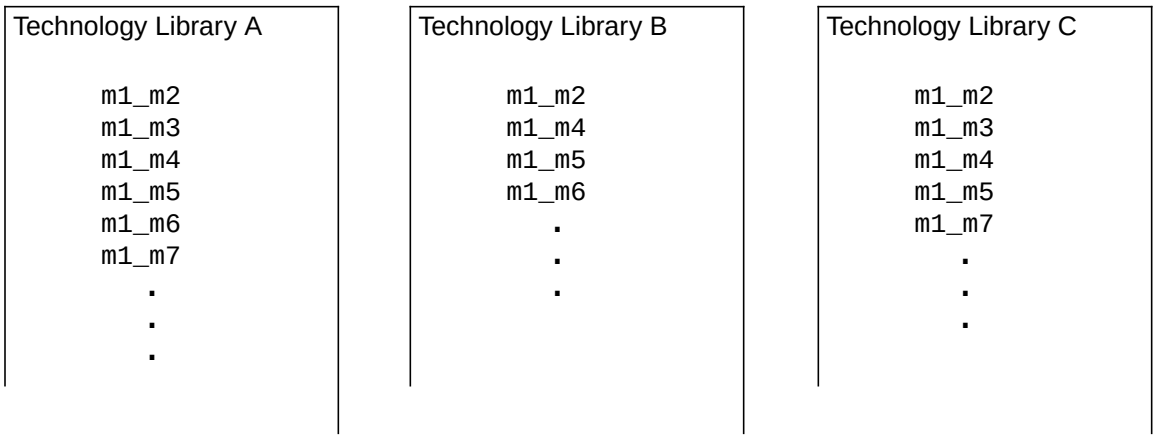
   From Library A: `m1_m3`, `m1_m7`

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`



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:

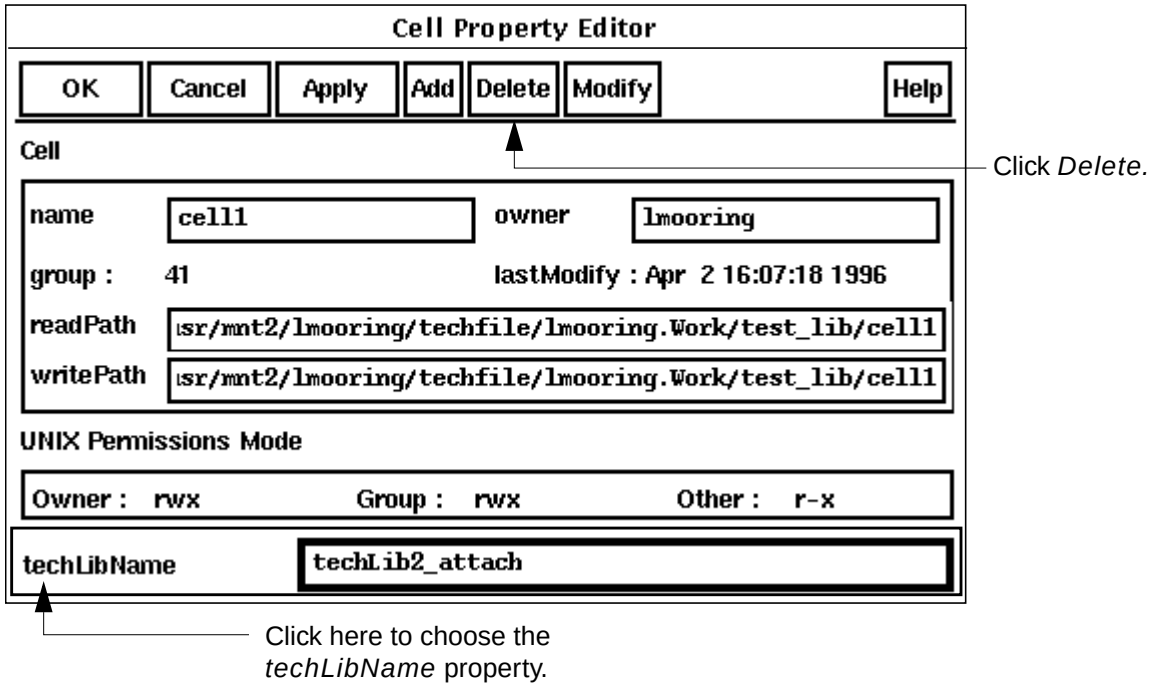
1. Reattach your design library to the original technology file (for example, Technology Library A).
2. Attach your design library to the new technology file (for example, Technology Library C).

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 with DFII, see "Setting Automatic Checkout and Checkin Preferences" in the *Design Framework II User Guide*.

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 Cadence Design Software Handles Multiple Display Resource Files" on page 33.

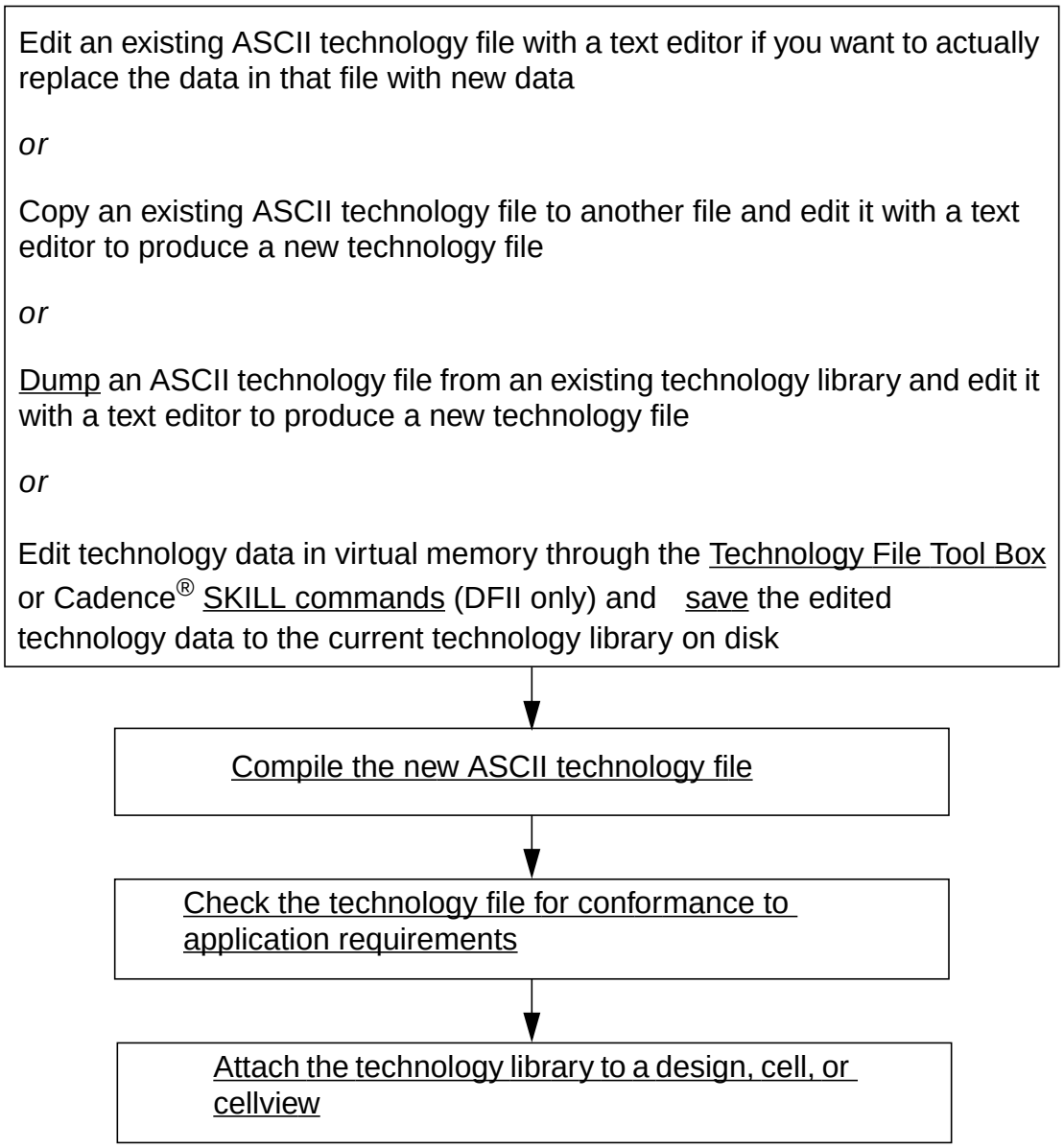
8

Editing, Reusing, and Merging Technology File Data

This chapter discusses the following:

The Technology File Updating Process

The following summarizes the general process for updating or creating a new technology file:



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 Technology File Tool Box.

  Note: Not all technology file data is accessible through the Technology File Tool Box. For classes of data that you cannot edit this way, you must use one of the other methods. See "Class Data Accessible through the Technology File Tool Box" on page 192 for details.

- Design Framework II (DFII) only: Use Cadence SKILL language functions to load a binary technology file and update it in memory.

  See "Editing Class Data with SKILL Functions (Design Framework II Only)" on page 207 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:

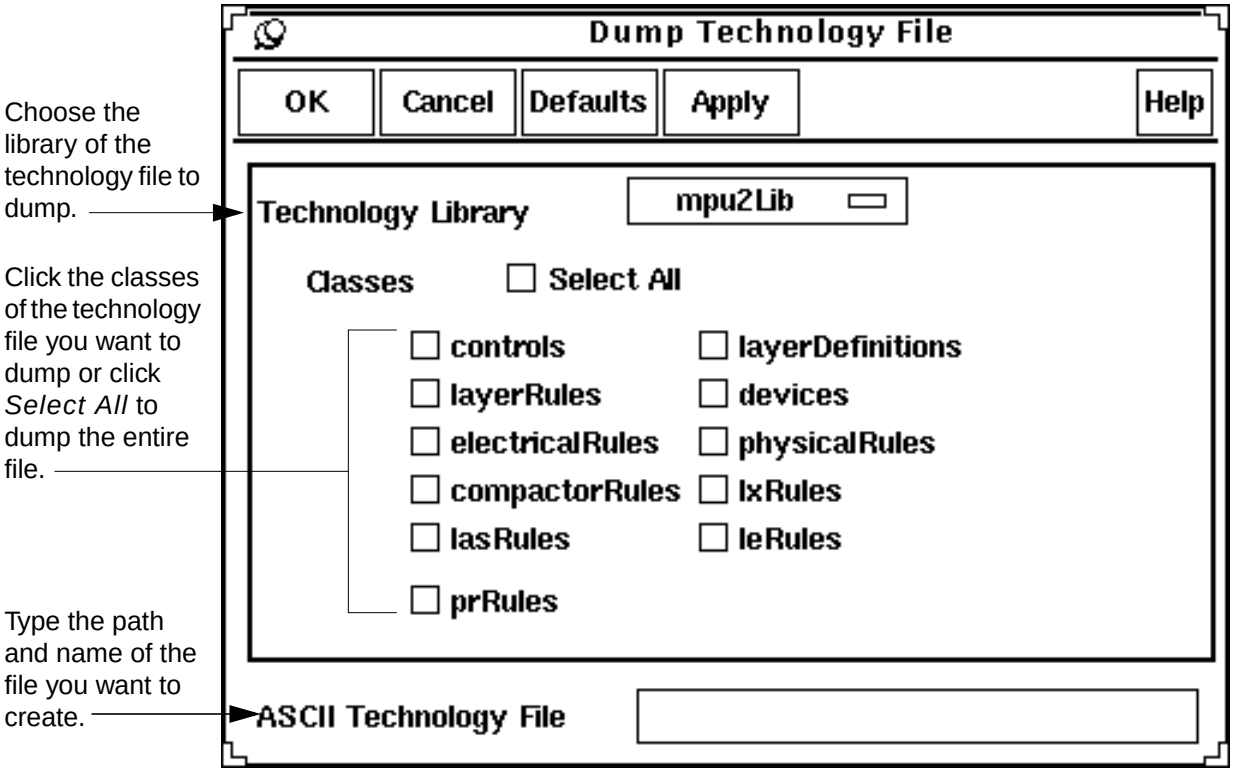
- It does not contain comments identified by semicolons (`;`) in the original ASCII file, although it does contain comments made with the `comment` statement.

- 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 Technology File Tool Box, choose *Dump*.

   The Dump 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 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.

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 of the technology file, refer to chapters 2 through 5 of this user guide and the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

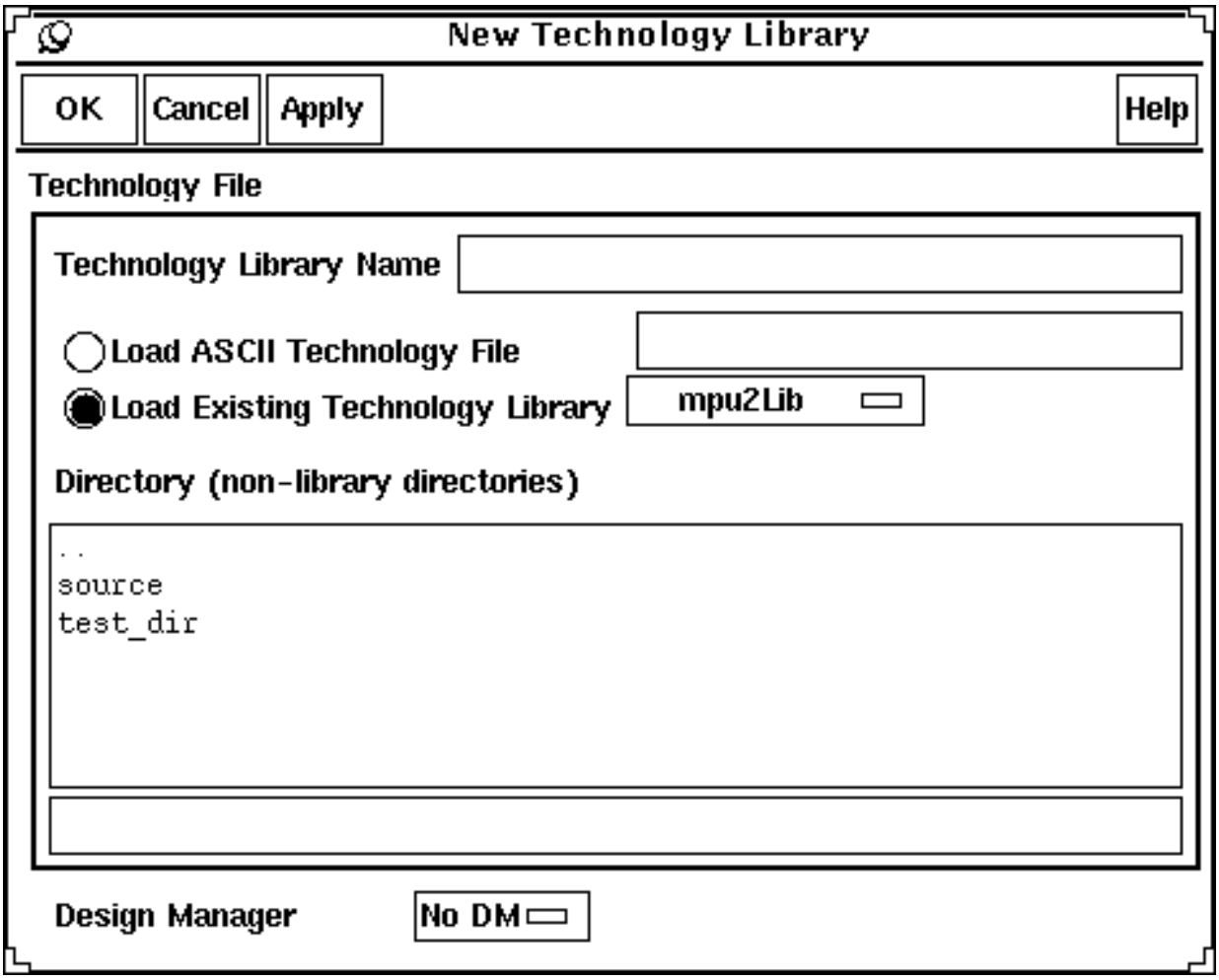
You can dump, edit, and load the ASCII file until you are satisfied with your changes. (For information on loading the technology file and merging its technology data with the technology data already in the technology library in virtual memory, see "Merging New Technology Data into an Existing Technology Library" on page 187. 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 189.) 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 Technology File Tool Box, choose *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

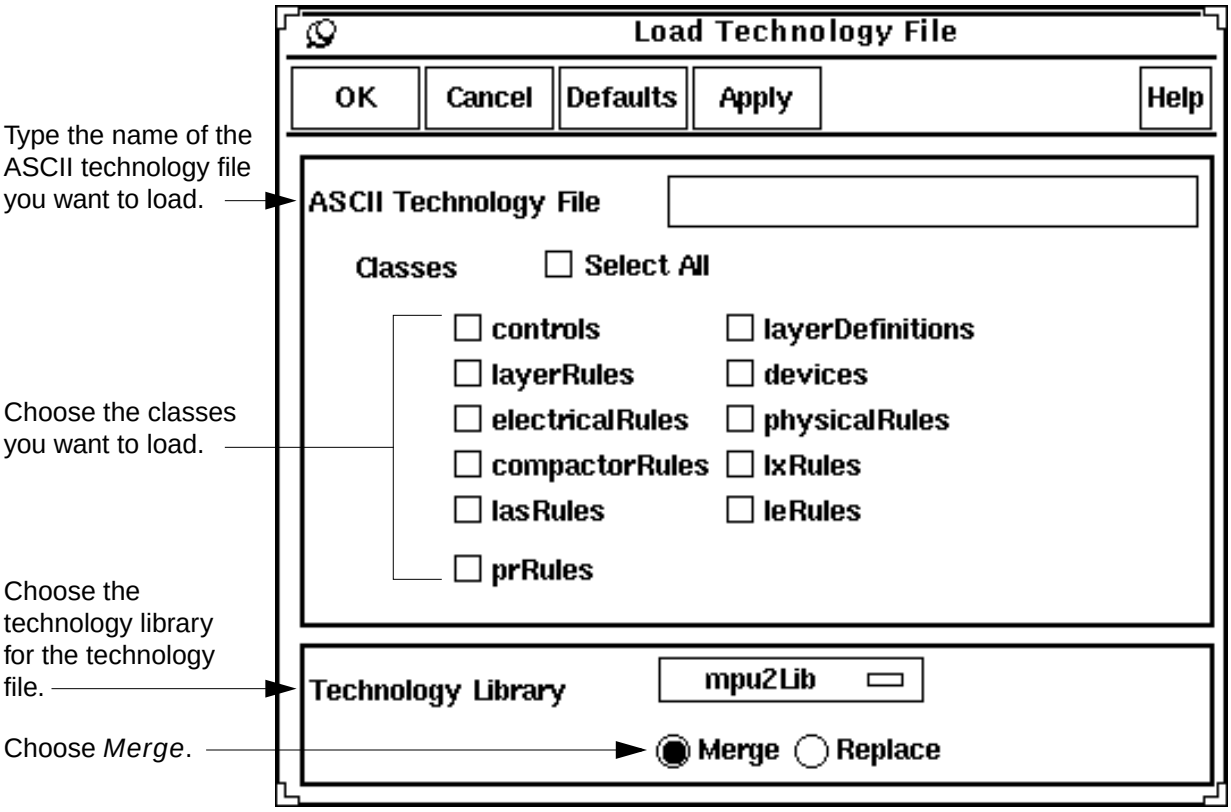
This section explains how to load data into virtual memory in two ways. With the *Technology File – Load* command, you can merge technology data in a technology file with the technology data already in virtual memory or you can replace the technology data in virtual memory with different technology data 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 189 for information on replacing the technology library):

1. From the Technology File Tool Box, choose *Load*.

The Load Technology File form appears.



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*.

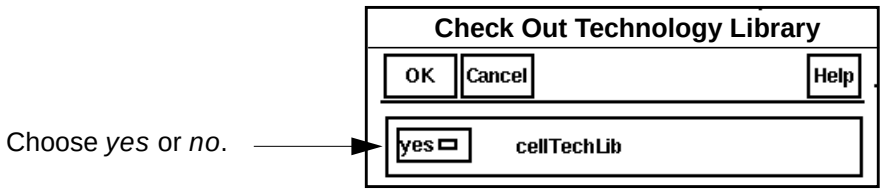
The software compiles the technology file and loads it into virtual memory. If your ASCII file does not contain all of the classes you chose, a dialog box appears listing the missing classes. Click *OK* to continue or *Cancel* to quit.

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 with DFII, see "Setting Automatic Checkout and Checkin Preferences" in the *Design Framework II User Guide*.

7. If you are prompted to check out the technology file, choose a check-out option and click *OK*.



- 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 DFII CIW and in the `techManager.log` file. For example:

```
Compiling class 'layerDefinitions'....
Compiling class 'devices'....
Compiling class 'physicalRules'....
Compiling class 'compactorRules'....
Technology file '~/tutorial.tf' loaded successfully.
```

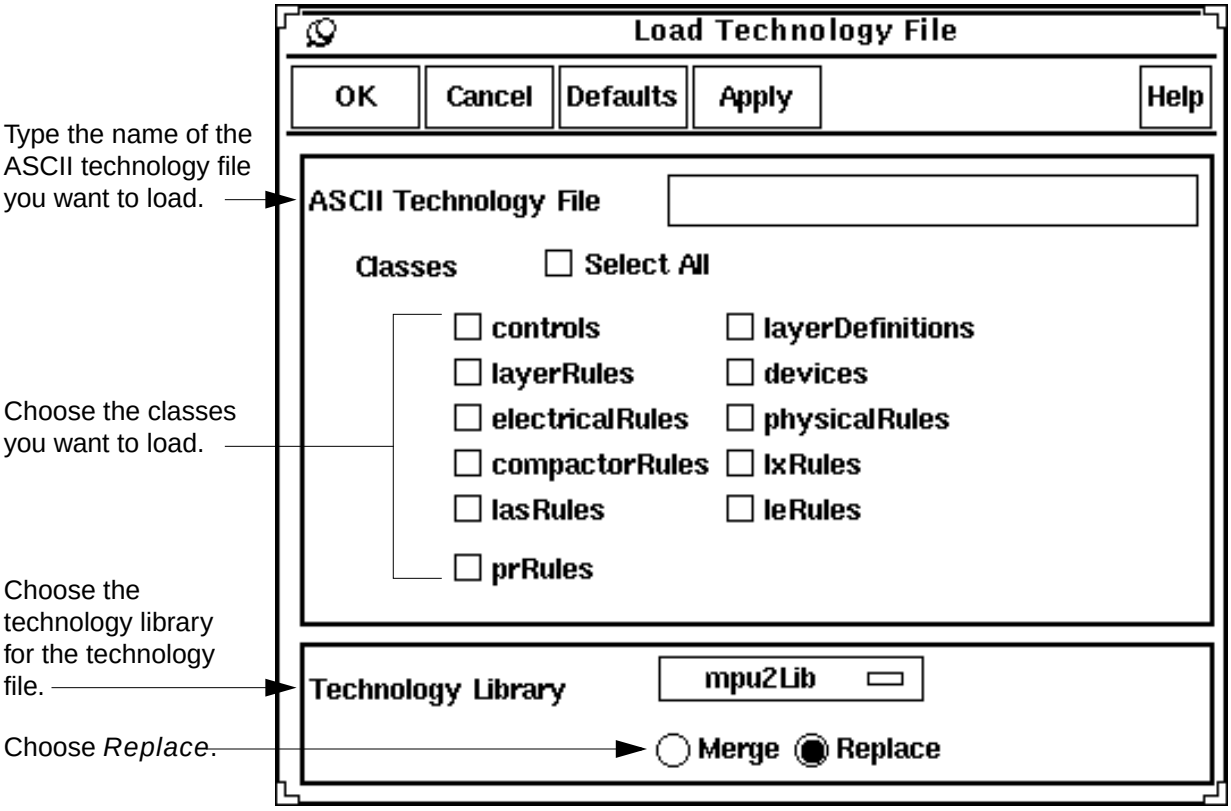
Replacing an Existing Technology File in a Technology Library

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 187 for information on merging the technology data with the existing technology library):

1. From the Technology File Tool Box, choose *Load*.

   The Load Technology File form appears



   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 *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*.

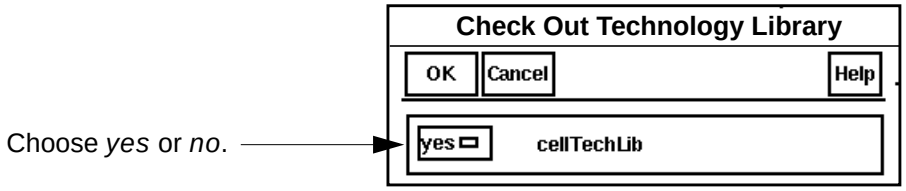
   The software compiles the technology file and loads it into virtual memory. If your ASCII file does not contain all of the classes you chose, a dialog box appears listing the missing classes. Click *OK* to continue or *Cancel* to quit.

   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 check-out options with DFII, see "Setting Automatic Checkout and Checkin Preferences" in the *Design Framework II User Guide*.

7. If you are prompted to check out the technology file, choose a check-out option and click *OK*.



   - 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 DFII CIW and in the `techManager.log` file. 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 Technology File Tool Box

You can edit most technology file data in virtual memory when you are running the design software. The Technology File Tool Box allows 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 of the technology file you can edit through the Technology File Tool Box and points you to the sections in this user guide containing more detailed information on each editing task.

Class Data Accessible through the Technology File Tool Box

The following table summarizes which classes and subclasses you can edit through the Technology File Tool Box and which you must edit with a text editor:

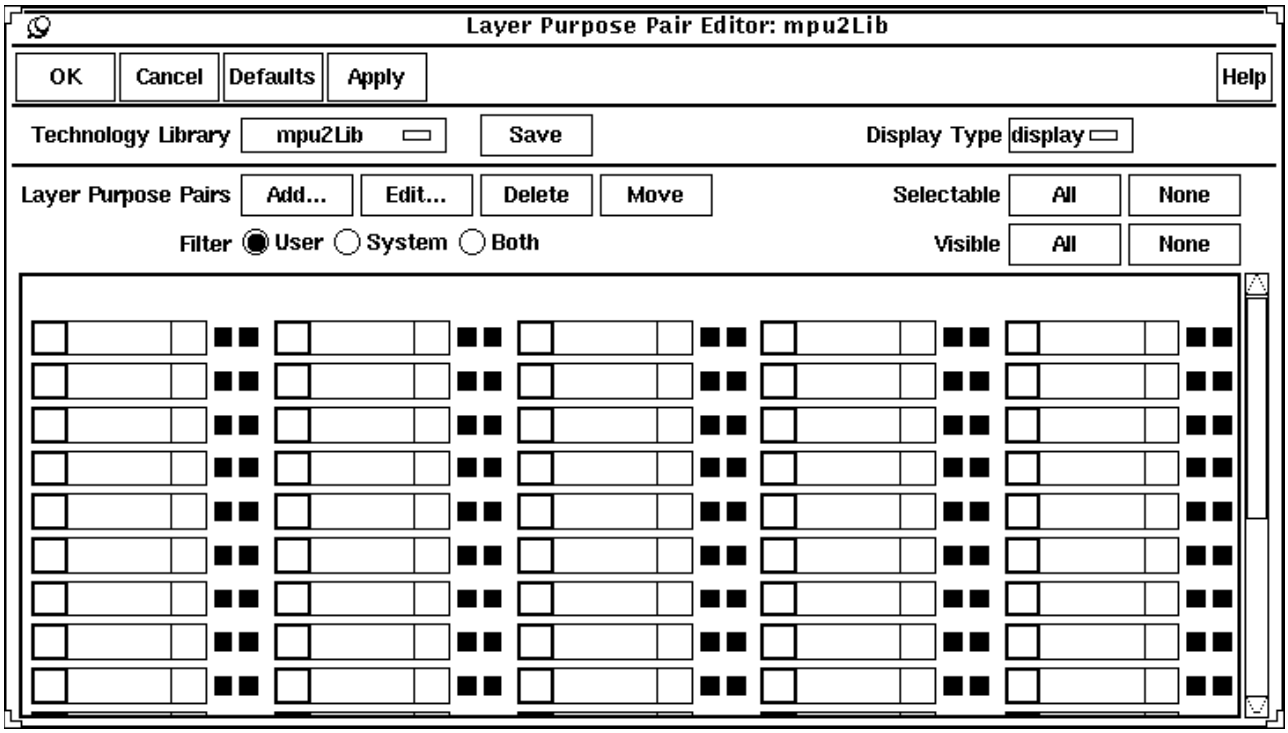
Class	Subclass	Tool Box Command
Controls	`techParams`	*Edit Rules*
	`techPermissions`	Not accessible; must edit with a text editor
Layer Definitions	All	*Edit Layers*
Devices	All	Not accessible; must edit with a text editor
Layer Rules	All	*Edit Rules*
Physical Rules	All	*Edit Rules*
Electrical Rules	All	*Edit Rules*
Layout Editor Rules	All	*Edit Rules*
Virtuoso® XL Rules	All	*Edit Rules*
Virtuoso Compactor Rules	All	*Edit Rules*
Place and Route Rules	All	*Edit Rules*

The Technology File Tool Box Commands

Two commands in the Technology File Tool Box access forms that allow you to edit class data in your technology file in virtual memory. They are

- *Edit Layers*
- *Edit Rules*

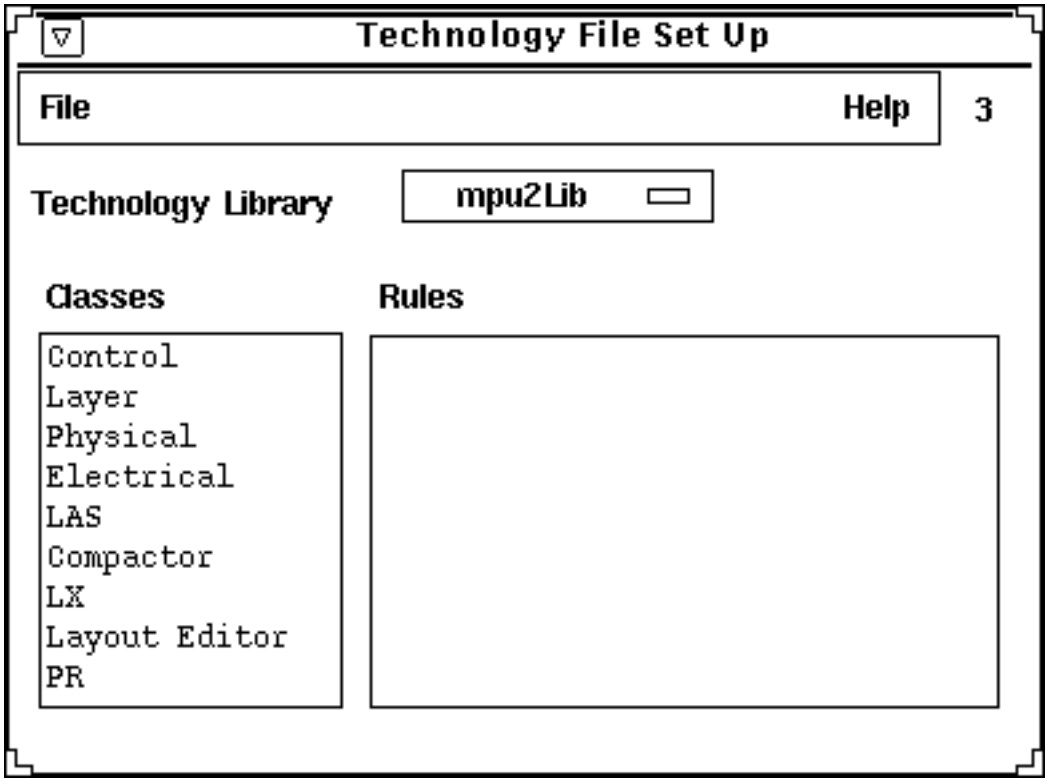
Edit Layers brings up the Layer Purpose Pair Editor form, in which you can edit, delete, and add layer definitions.



For a description of this form, see Appendix A.

For detailed information on editing layer definitions, refer to Chapter 9, “Editing Class Data through the Technology File Tool Box: Controls and Layer Definitions.”

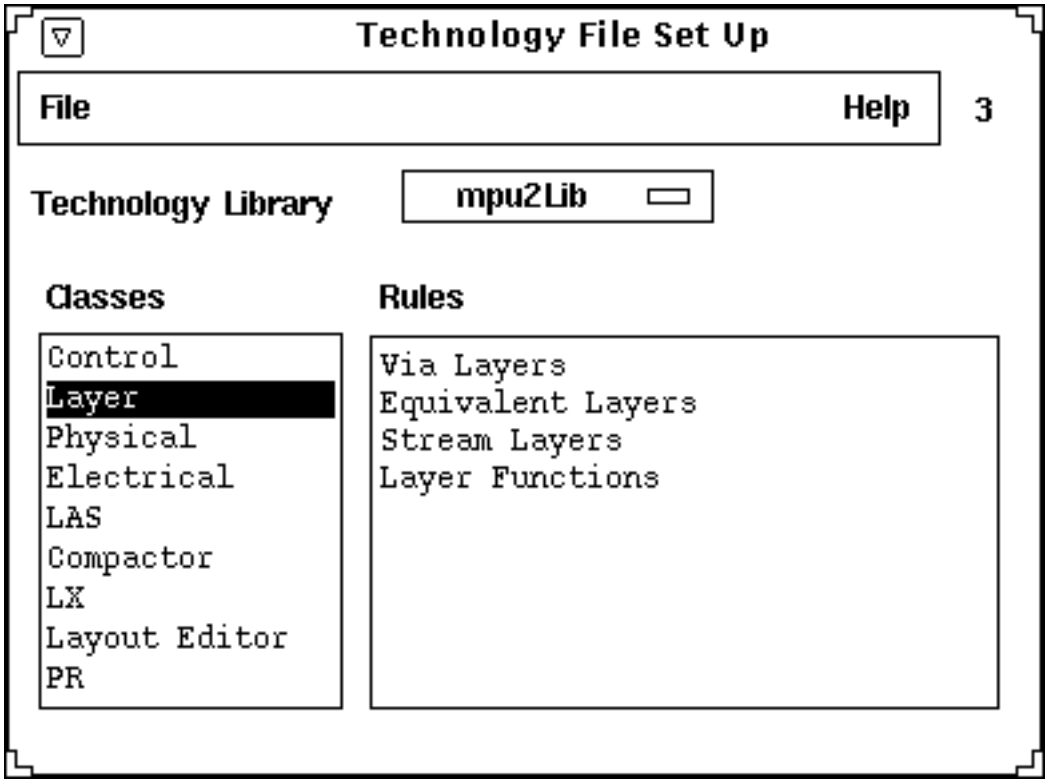
Edit Rules brings up the following initial Technology File Set Up form:



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 the Technology File Tool Box: 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 the Technology File Tool Box: Generic Rules.”

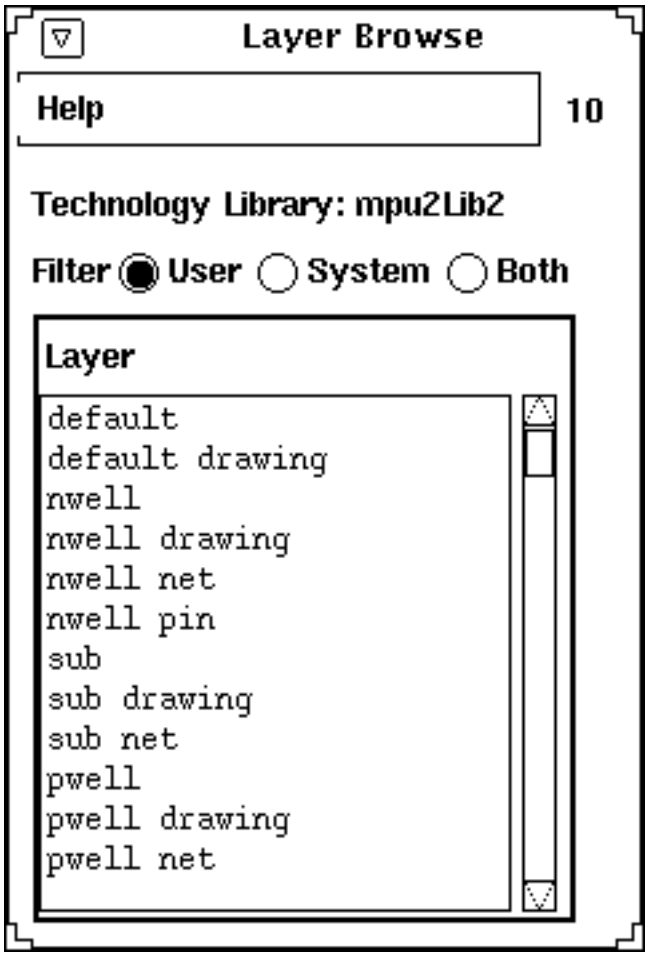
For detailed information on editing application-specific rules (Virtuoso Layout Editor rules, Virtuoso XL Layout Editor (Virtuoso XL) rules, and Virtuoso Compactor rules), refer to Chapter 11, “Editing Class Data through the Technology File Tool Box: Application-Specific Rules.”

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.

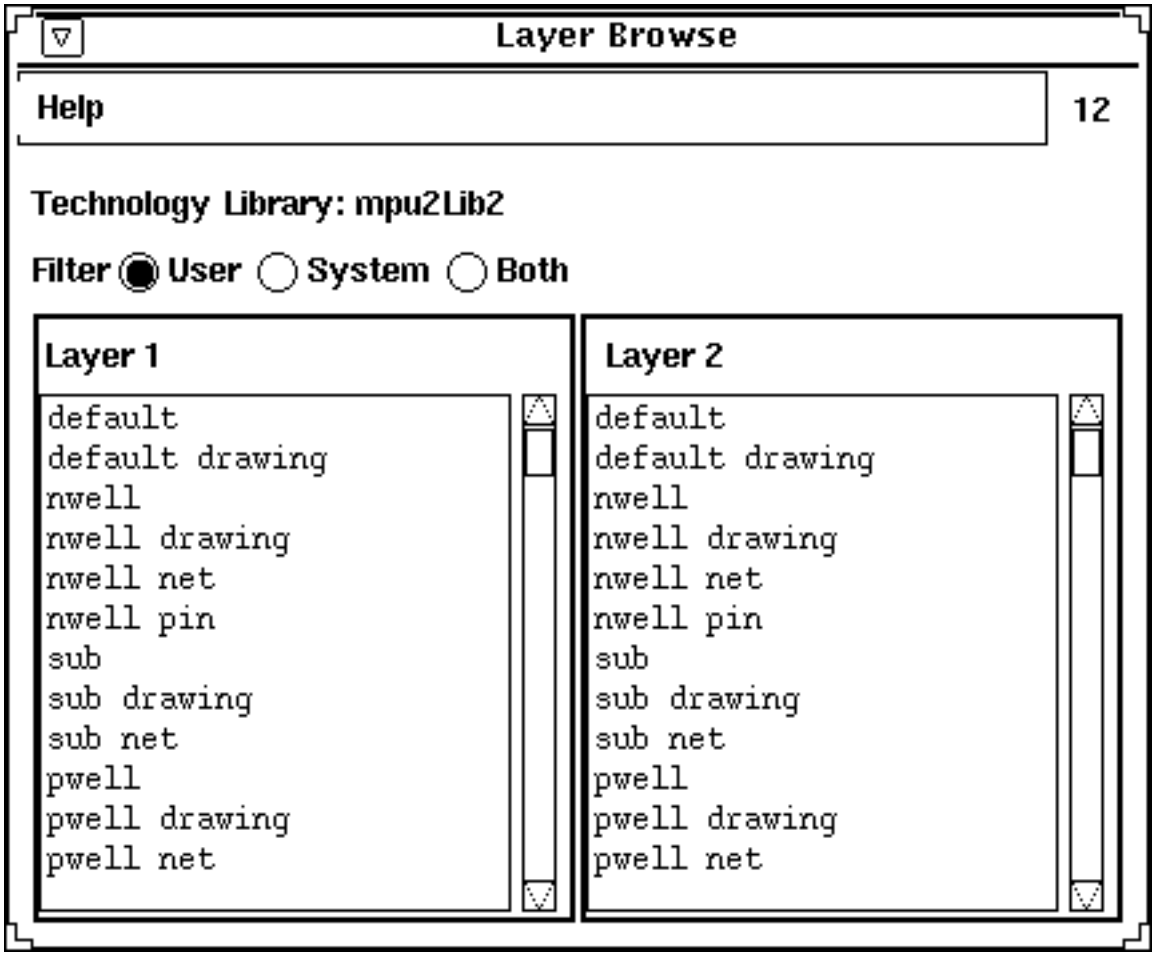
Sample Layer Browser Forms

The following is a sample Layer Browser form for rules that use a single layer:



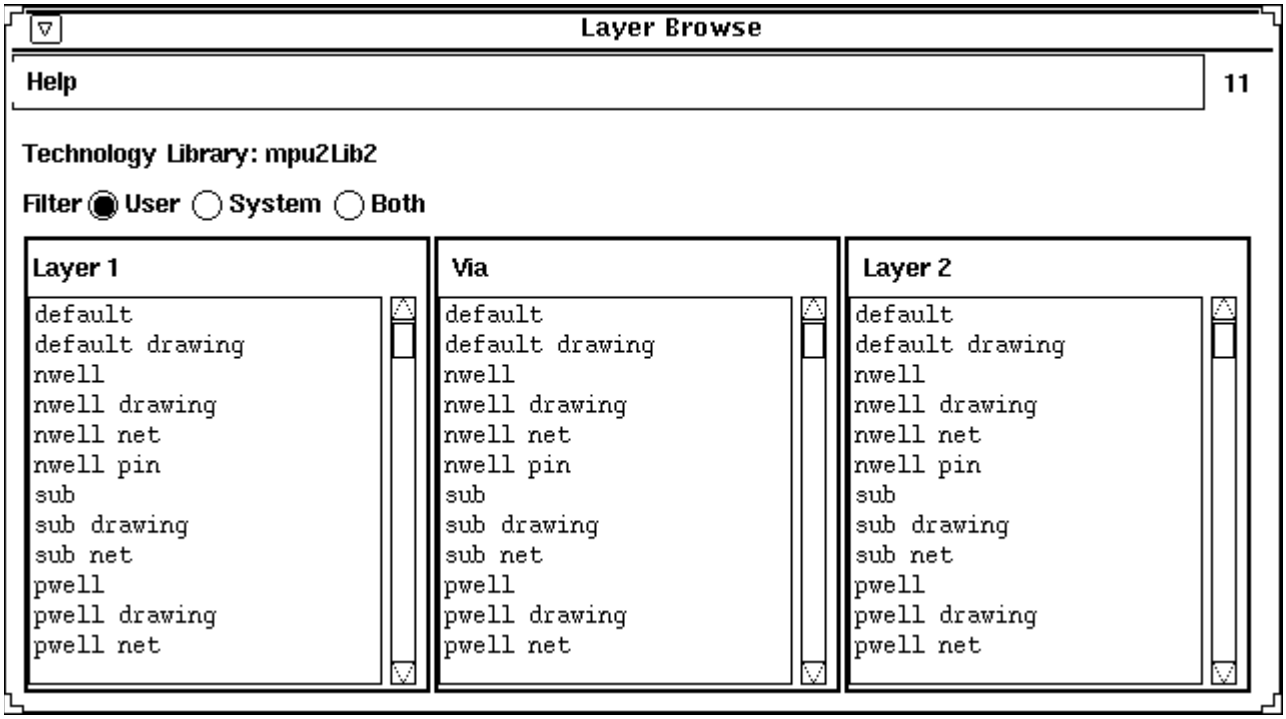
For a description of this form, see Appendix A.

The following is a sample Layer Browser form for rules that use two layers:



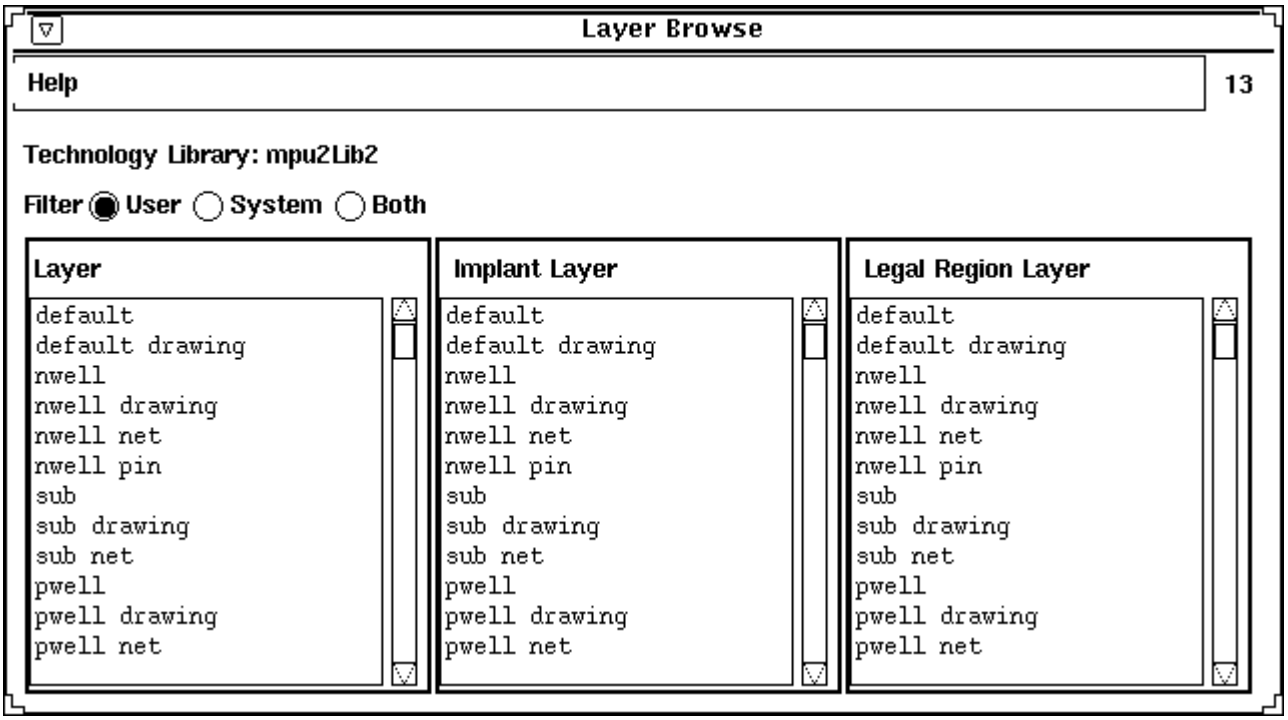
For a description of this form, see Appendix A.

The following is a sample Layer Browser form for the `viaLayers` Layer Rules subclass:



For a description of this form, see Appendix A.

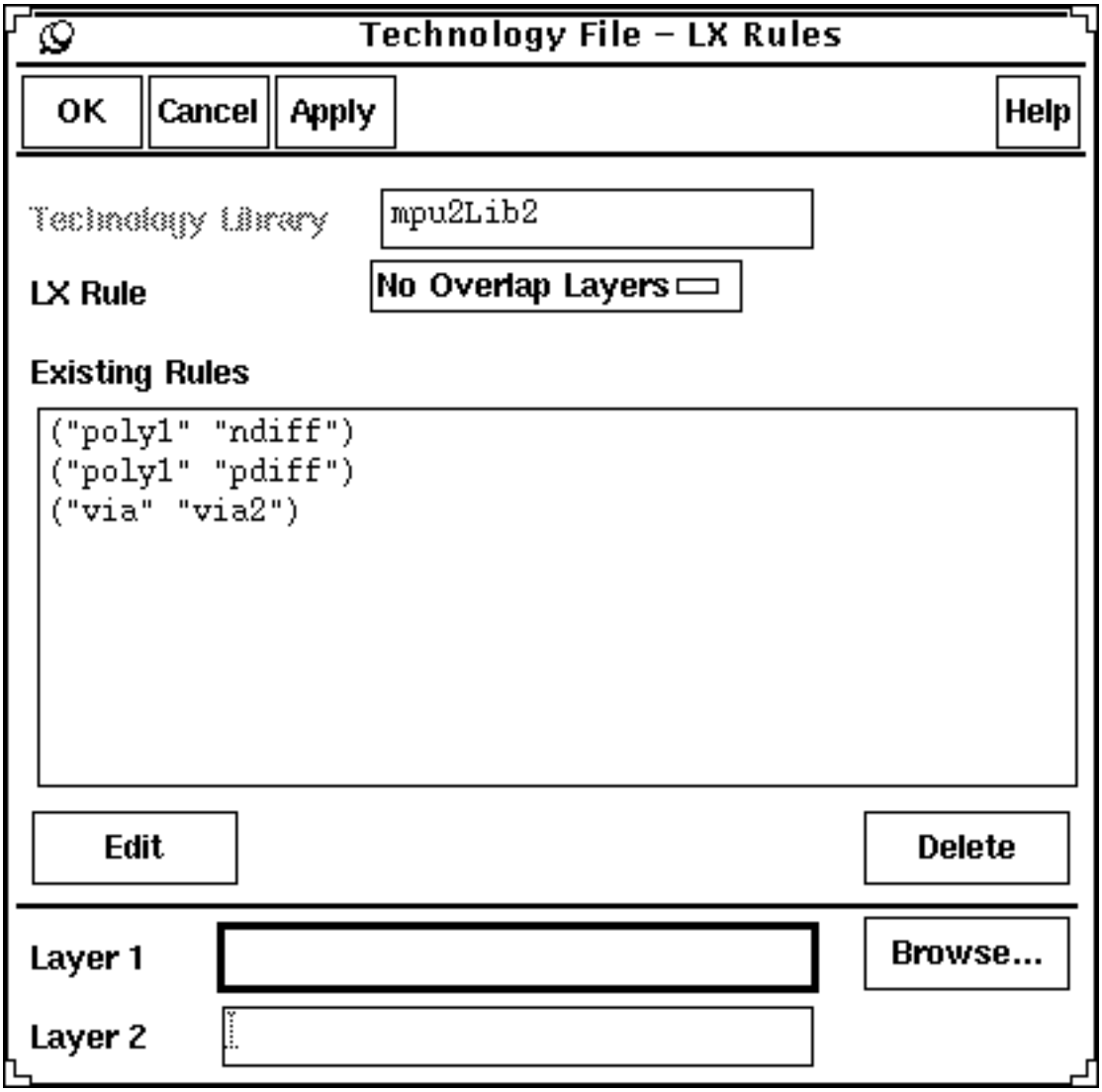
The following is a sample Layer Browser form for the `symWires` Compactor Rules subclass:



For a description of this form, see Appendix A.

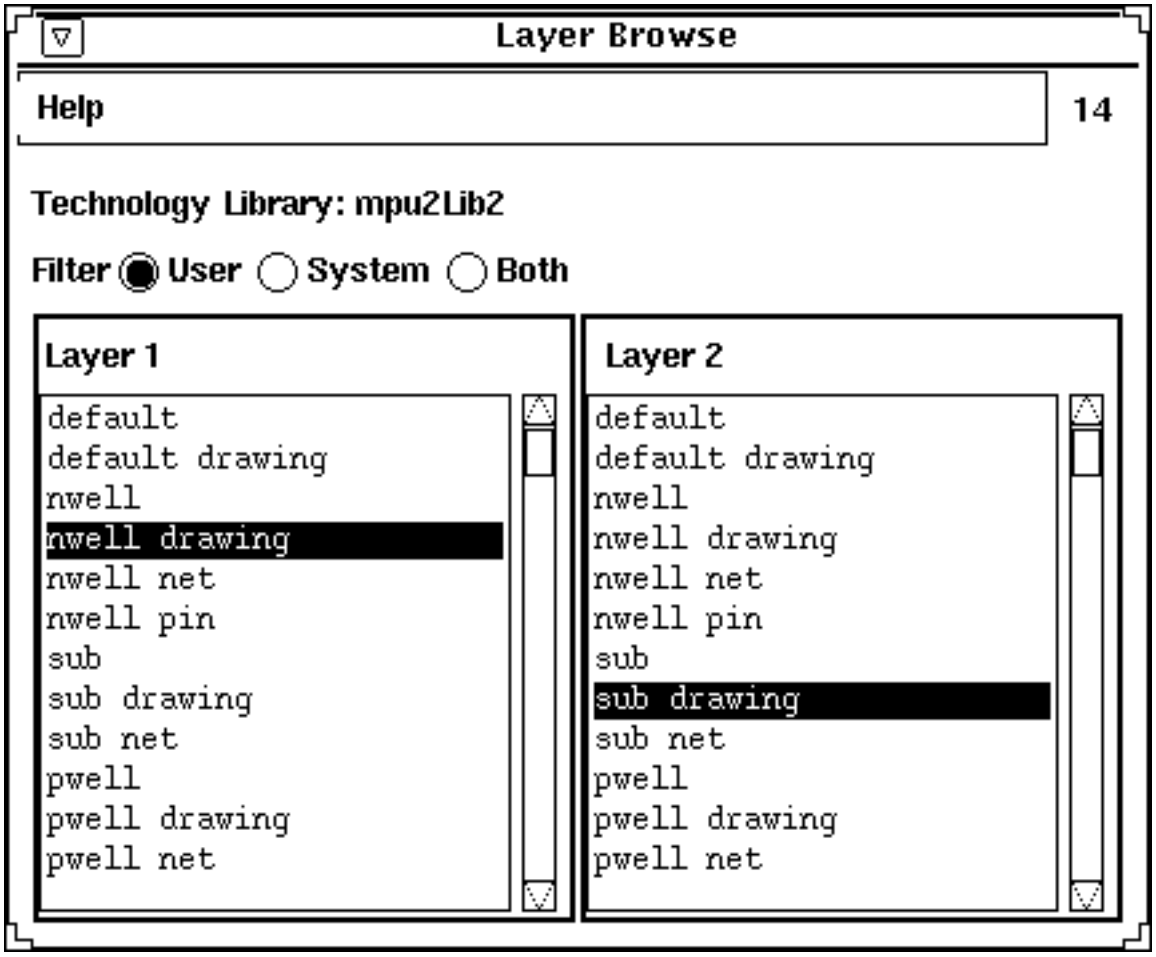
Example of Using a Layer Browser

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:

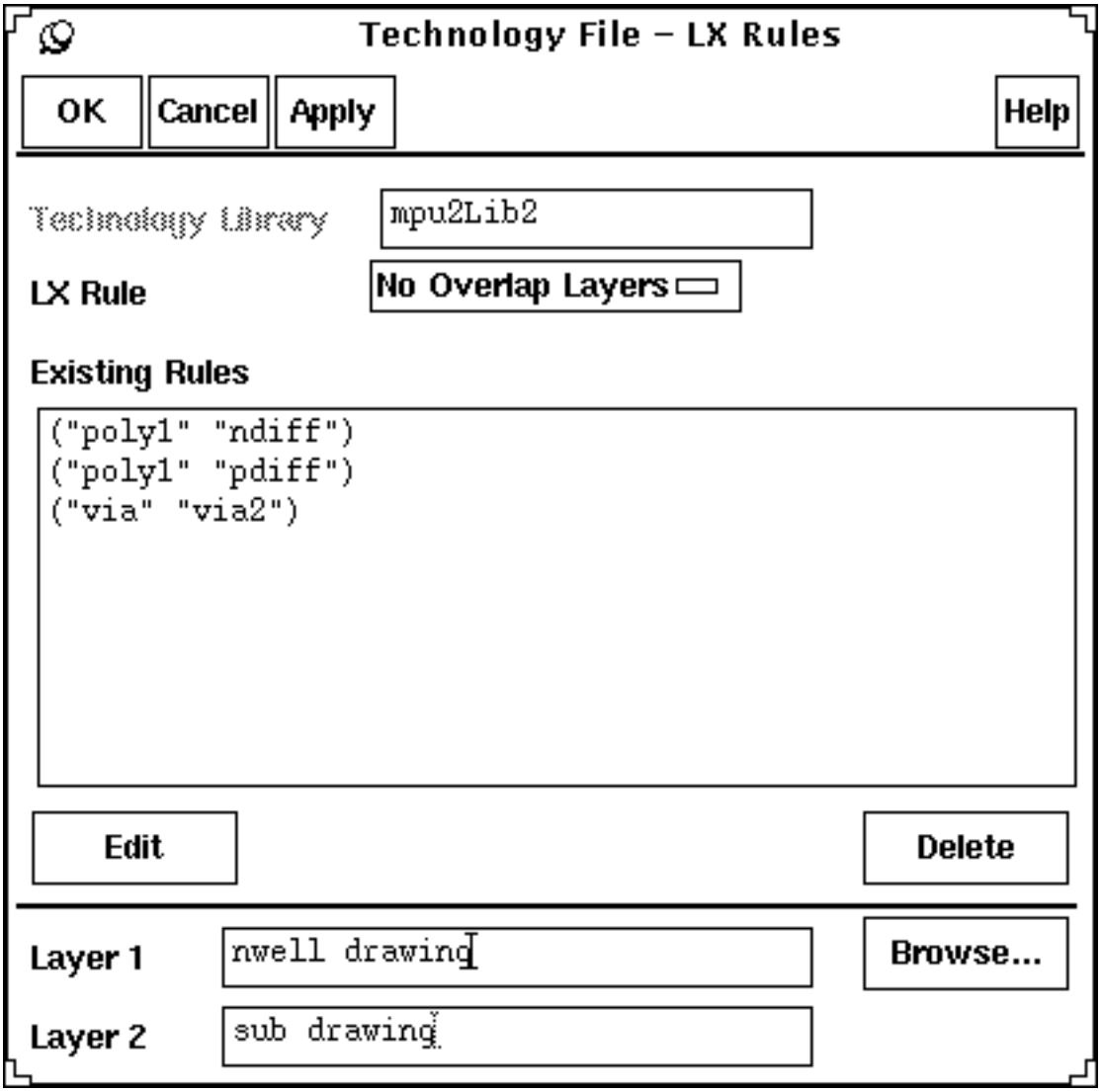


When you choose *Browse* on this form, the software displays the Layer Browser. As you click on layers in this Layer Browser, the software highlights them in the browser and automatically

loads the layer names into the appropriate fields on the Set Up form. The following illustrates choosing layers in the Layer Browser:



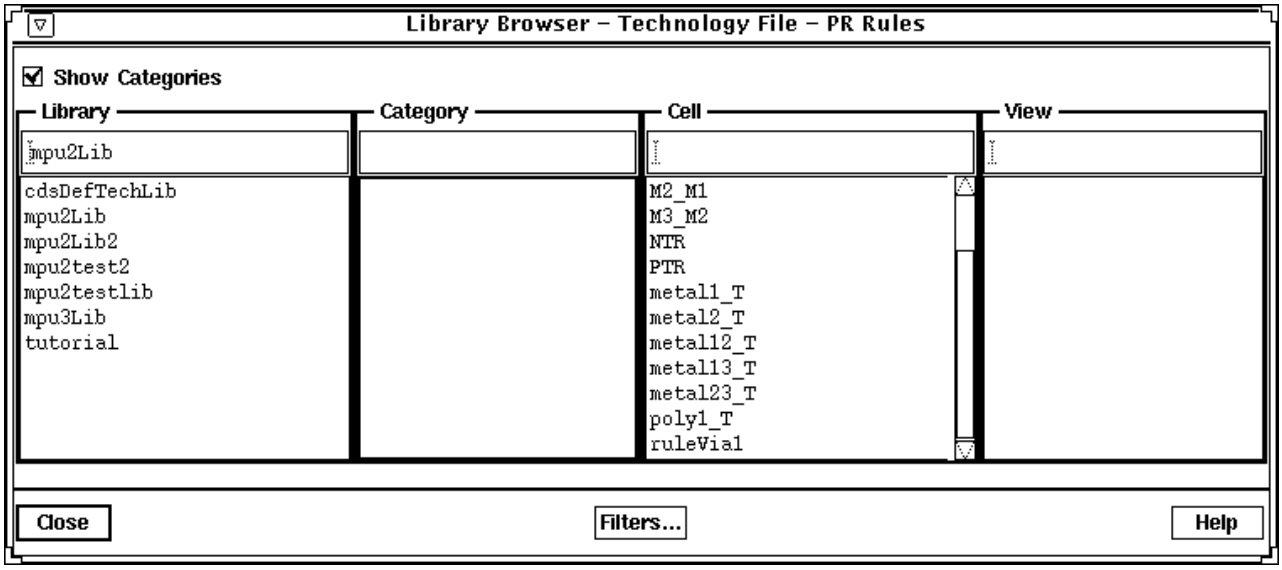
The following illustrates how the software loads the layers selected in the Layer Browser into the *Layer* fields on the Set Up form:



For details on using the various Set Up forms, see chapters 9, 10, and 11.

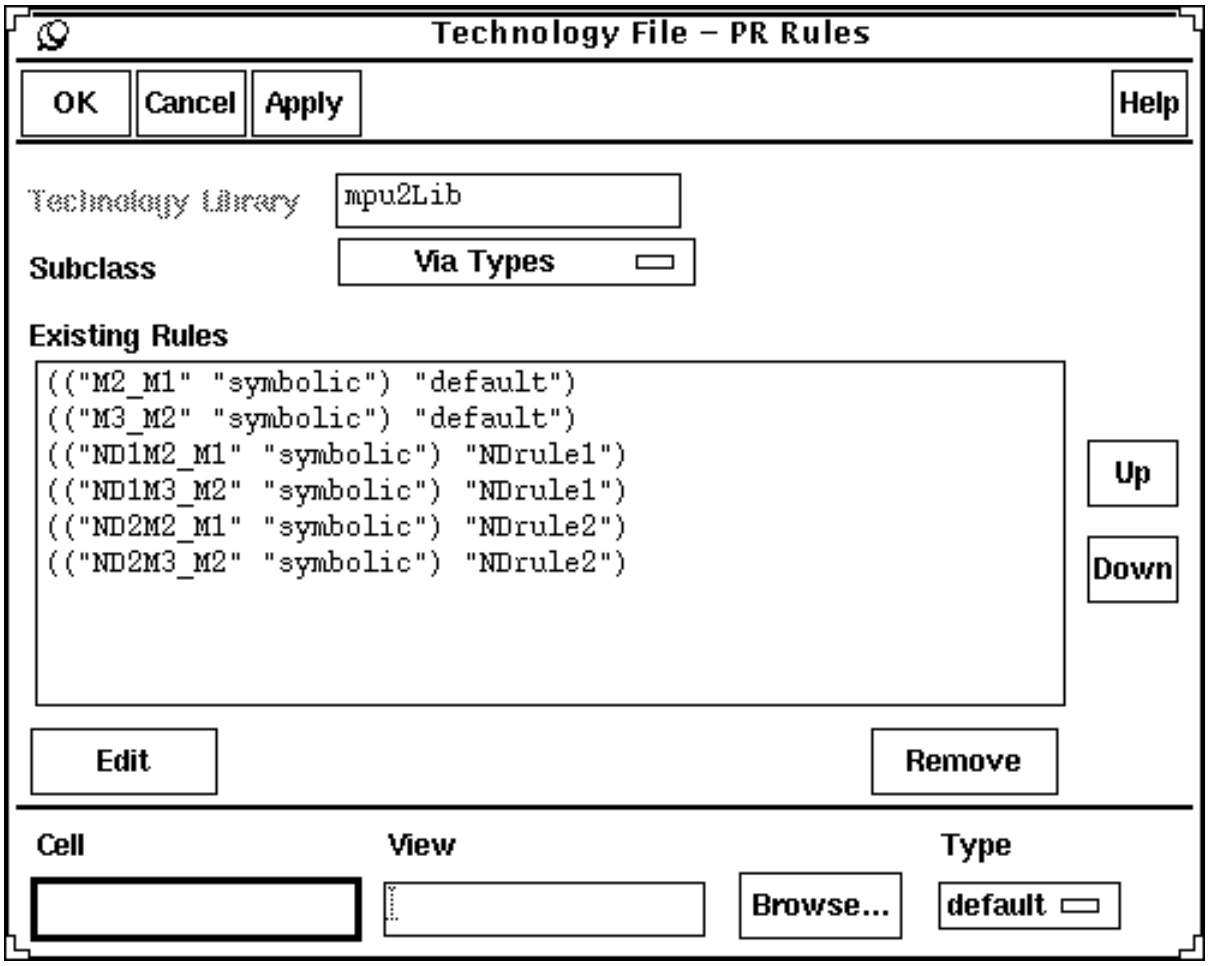
Library Browsers

Some forms requiring that you supply library elements contain a *Browse* button that allows you to display a Library Browser from which you can browse through and choose existing library elements, such as cells and cellviews. The following is a sample Library Browser form:

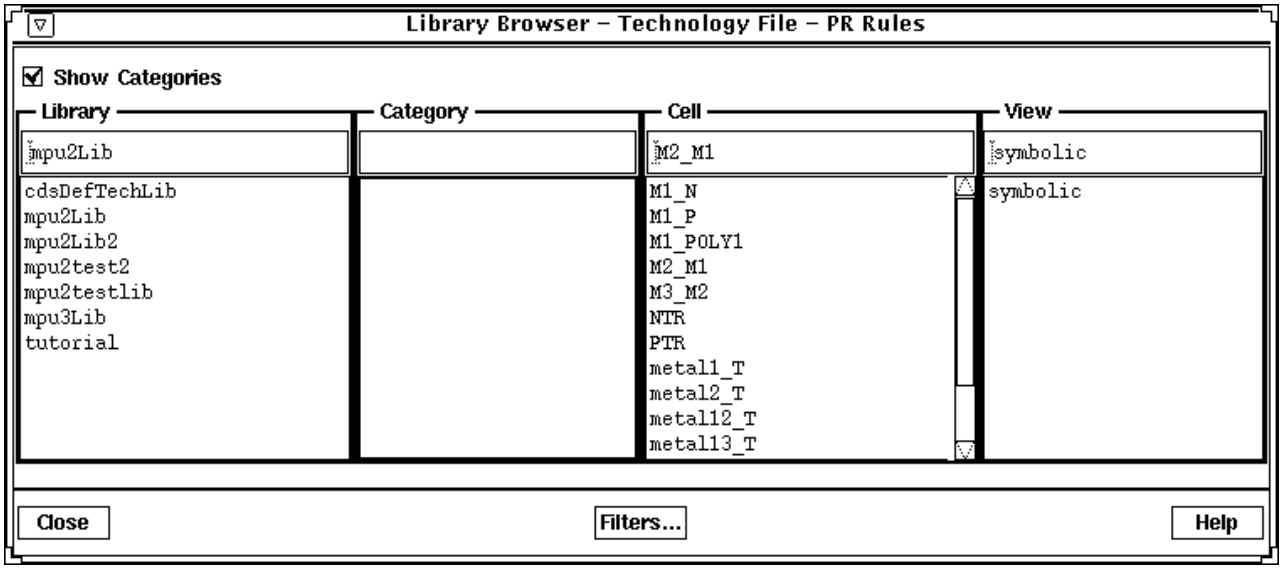


Example of Using a Library Browser

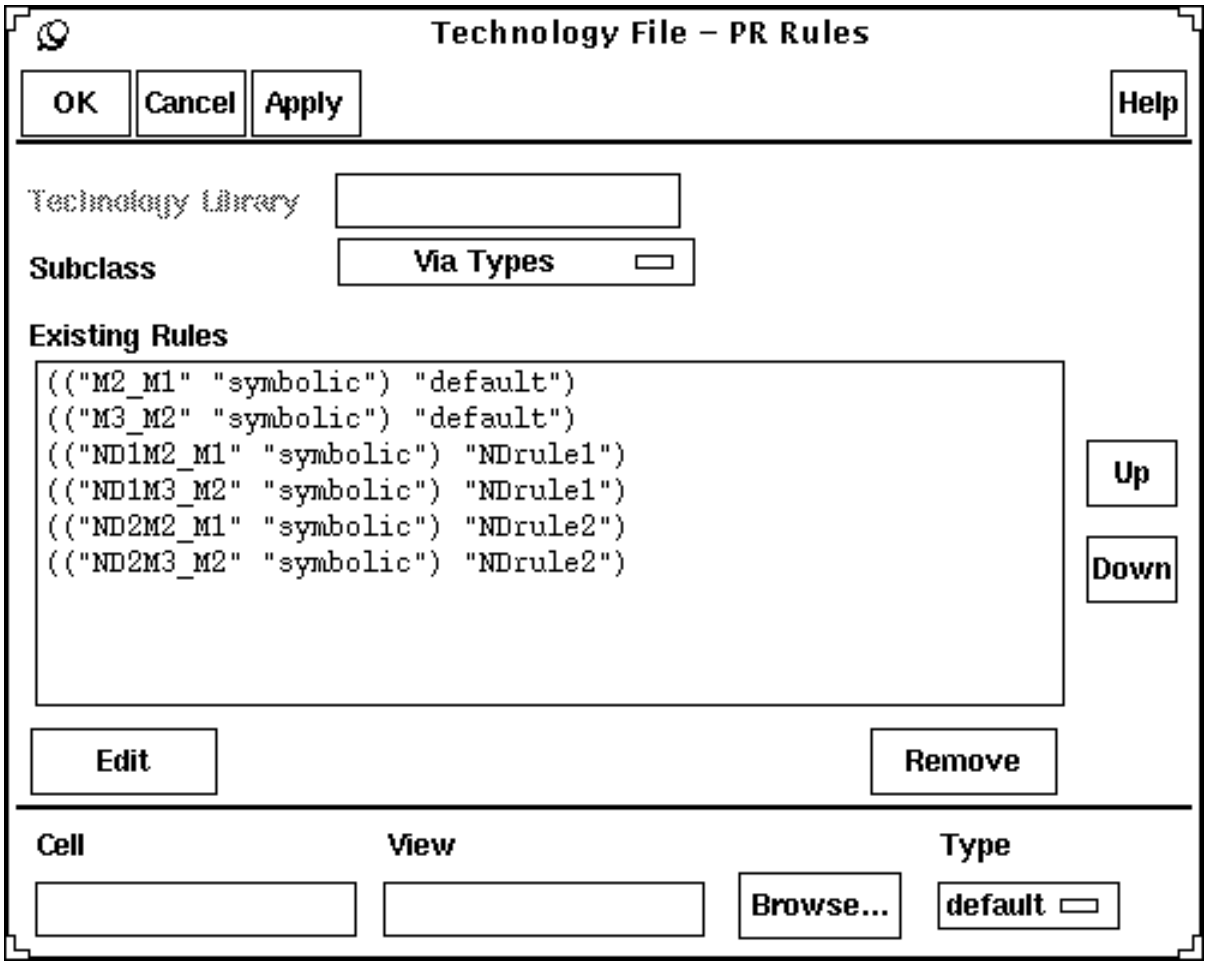
When you choose a cell in a Library Browser, the software automatically loads it into the appropriate field in the current Set Up form. For example, the following is the Set Up form for editing the Via Types subclass of the Place and Route class:



When you choose *Browse* on this form, the software displays the Library Browser. As you click on cells in this Library Browser, the software highlights them in the browser and displays the views for that cell; it also automatically loads the layer names into the appropriate fields on the Set Up form. The following illustrates choosing a cell in the Library Browser:



The following illustrates how the software loads the cell and view selected in the Library Browser into the *Cell and View* fields on the Set Up form:

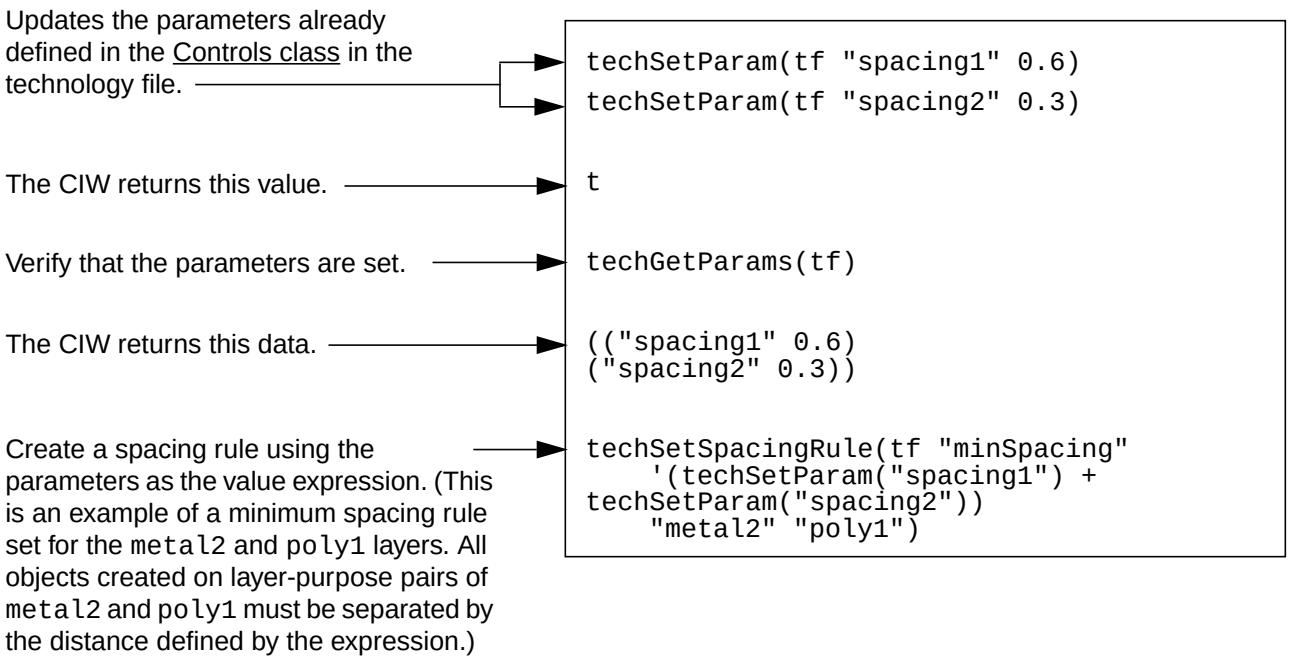

Technology File – PR Rules
OK
Cancel
Apply
Help
Technology Library
Subclass
Via Types
Existing Rules
(("M2_M1" "symbolic") "default")
(("M3_M2" "symbolic") "default")
(("ND1M2_M1" "symbolic") "NDrule1")
(("ND1M3_M2" "symbolic") "NDrule1")
(("ND2M2_M1" "symbolic") "NDrule2")
(("ND2M3_M2" "symbolic") "NDrule2")
Up
Down
Edit
Remove
Cell
View
Type
Browse...
default

Editing Class Data with SKILL Functions (Design Framework II Only)

The Controls Class

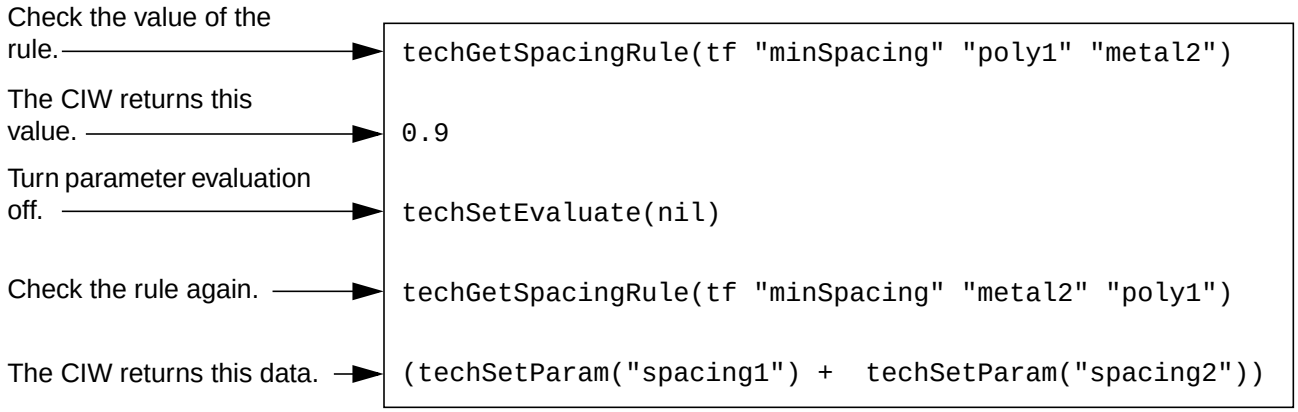
Adding a Control Parameter

To add a parameter to the `techParams` subclass 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:



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.



Remember to turn parameter evaluation back on. When it is off, none of your expressions or parameters are evaluated; instead, they are read as strings.

The Layer Definitions Class

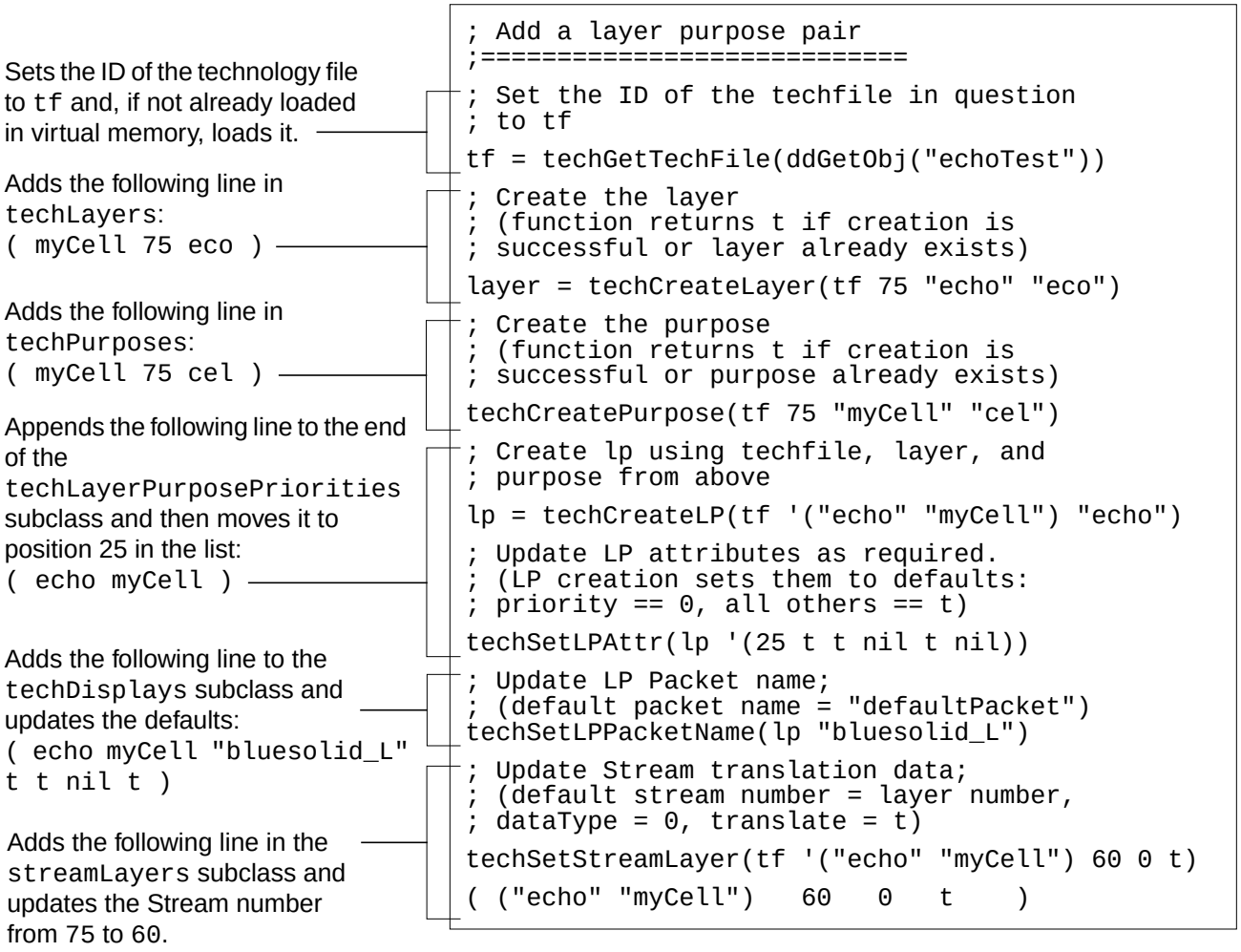
Adding a Layer

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:

SKILL Function	Description
`techCreateLP`	Creates a layer with default priority and display attributes
`techSetLPAttr`	Updates the default values of all the display attributes except packet
`techSetLPPacketName`	Updates the default packet assigned to the layer
`techSetStreamLayer`	If necessary, modifies the default Stream data created by `techCreateLP`

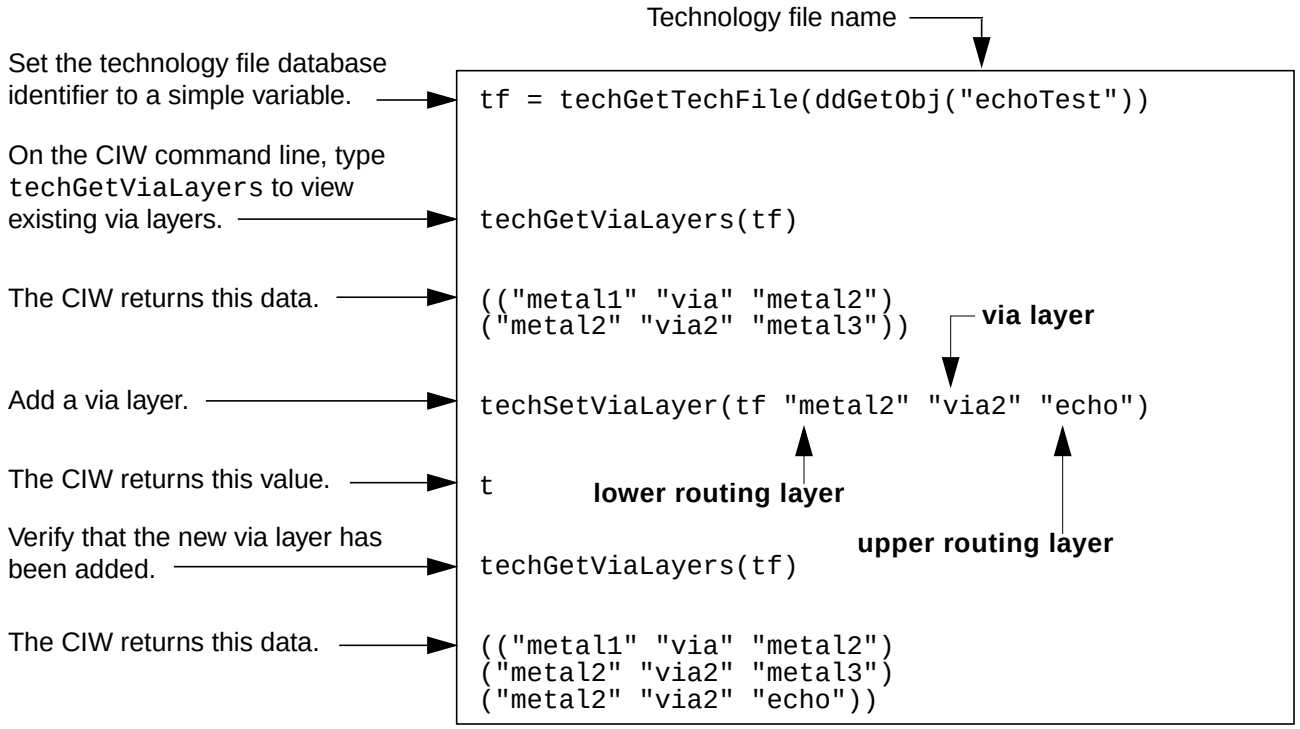
The following is a portion of a script that creates a layer.



The Layer Rules Class

Adding Via Layers

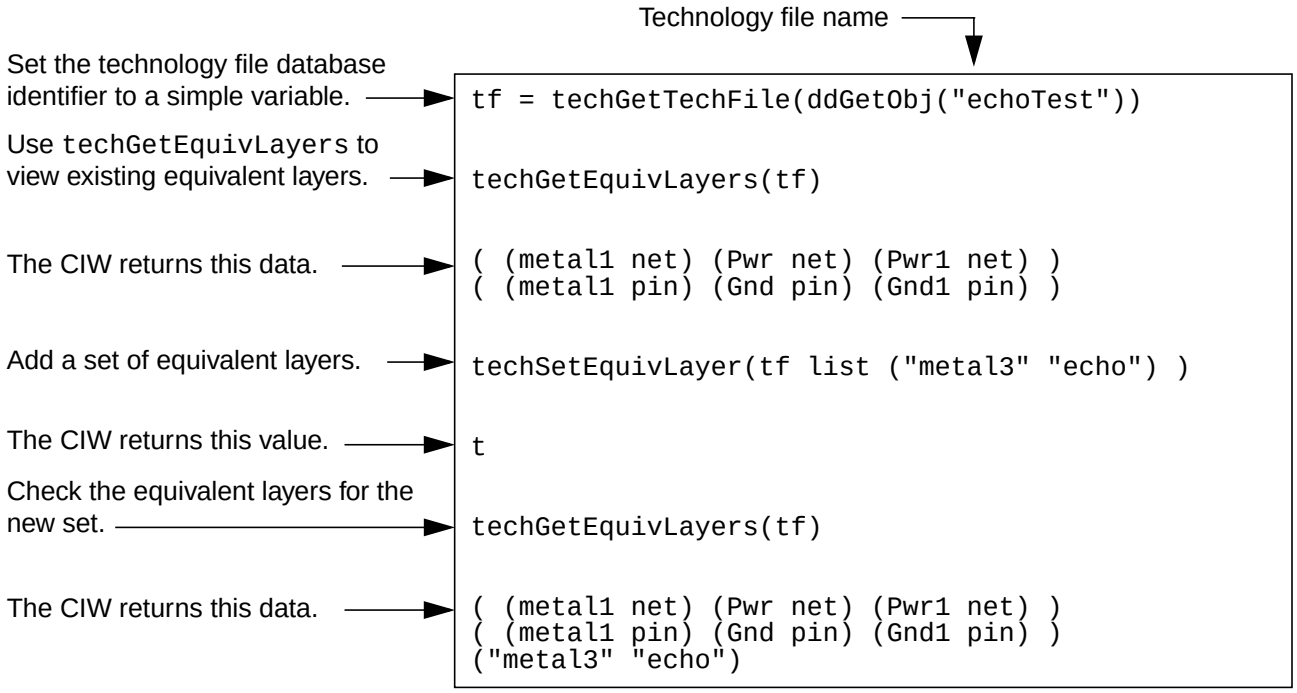
To add a via layer to the `viaLayers` subclass of a technology file in virtual memory with SKILL functions, type the `techSetViaLayer` SKILL function in the CIW command line. If the `viaLayers` subclass does not exist, this function creates it. The following is part of an interactive session in the CIW that adds a via layer to the technology file in virtual memory.



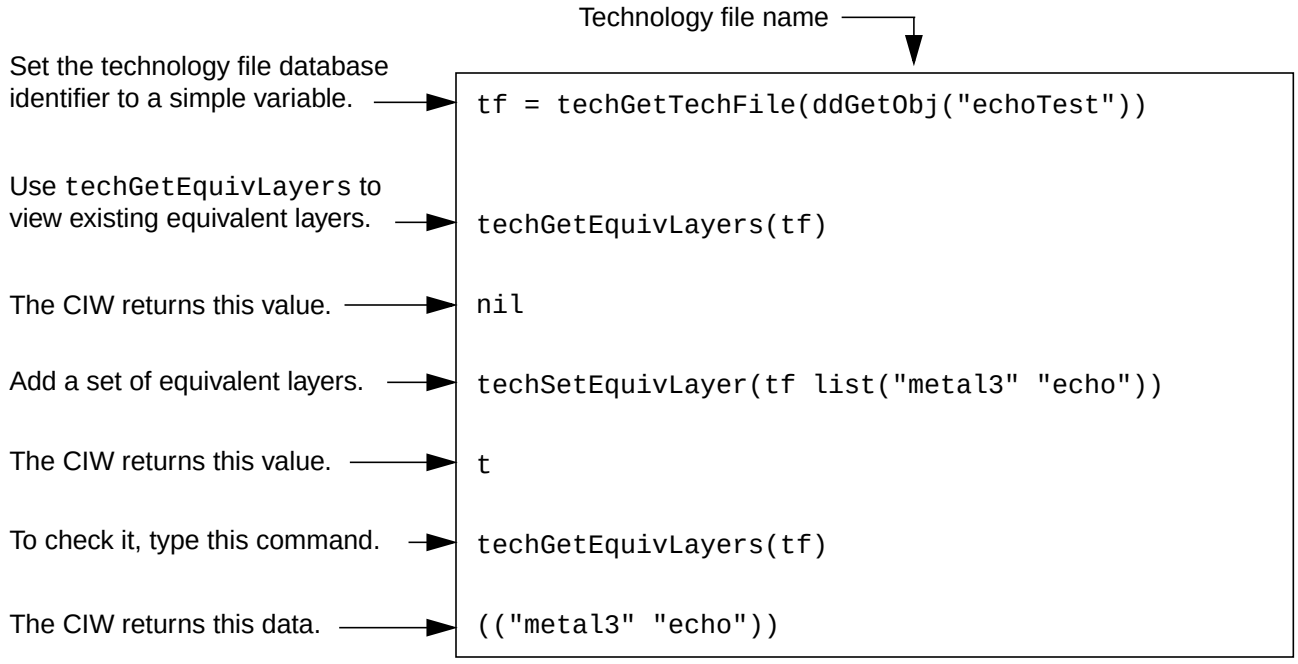
Adding Equivalent Layers

To add a set of equivalent layers to the `equivalentLayers` subclass 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.

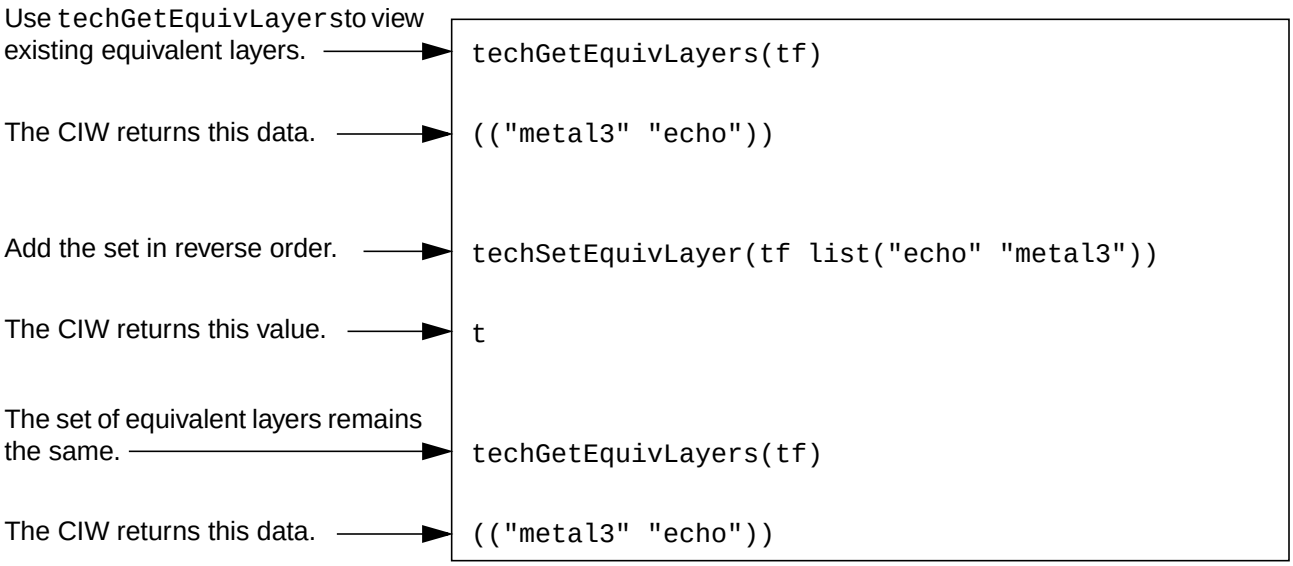
The following is an example of the functions you can type into the CIW and the results you get:



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.



The order of the layers is not important. If you try to add a duplicate set, the function returns `t` but does not add an additional set.



Adding Stream Layers

To add Stream translation data to the `streamLayers` subclass 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.

```
; 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 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 (Design Framework II Only)" on page 207 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 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:

SKILL Function	Description
`techSetLeLswLayers()`	Overwrites the existing `leLswLayers` subclass with specified data
`techSetLeLswLayer()`	Appends the specified layer to the `leLswLayers` subclass
`techGetLeLswLayers()`	Returns the layers listed in the `leLswLayers` subclass
`techIsLeLswLayer()`	Returns `t` if the specified layer is listed in the `leLswLayers` subclass

The Virtuoso XL Rules Class

The following table lists the SKILL functions that operate on the Virtuoso XL Rules class of the technology file:

SKILL Function	Description
`techSetLxExtractLayers()`	Overwrites the existing `lxExtractLayers` subclass with specified data
`techSetLxExtractLayer()`	Appends the specified layer to the list in the `dleExtractLayers` subclass

SKILL Function	Description
techGetLxExtractLayers()	Returns the layers listed in the dleExtractLayers subclass
techIsLxExtractLayer()	Returns t if the specified layer is listed in the dleExtractLayers subclass
techSetLxNoOverlapLayers()	Overwrites the existing lxNoOverlapLayers subclass with specified data
techSetLxNoOverlapLayer()	Appends the specified layer pair to the dleNoOverlapLayers subclass
techGetLxNoOverlapLayers()	Returns the layer pairs listed in the dleNoOverlapLayers subclass
techIsLxNoOverlapLayer()	Returns t if the specified layer pair is listed in the dleNoOverlapLayers subclass

The Virtuoso Compactor Rules Class

The following table lists the SKILL functions that operate on the Virtuoso Compactor Rules class of the technology file:

SKILL Function	Description
techSetCompactorLayers()	Overwrites the existing compactorLayers subclass with specified data
techSetCompactorLayer()	Appends the specified layer to the list in the compactorLayers subclass
techGetCompactorLayers()	Returns the layers listed in the compactorLayers subclass
techGetCompactorUsage()	Returns the Virtuoso Compactor keyword for the specified layer in the compactorLayers subclass
techIsCompactorLayer()	Returns t if the specified layer is listed in the compactorLayers subclass
techSetSymWire()	Appends the specified wire to the symWires subclass
techGetSymWires()	Returns the wires and wire definitions listed in the symWires subclass

SKILL Function	Description
techGetSymWireParams()	Returns the definition of the specified wire listed in the symWires subclass
techSetSymRules()	Appends the specified symbolic rule to the symRules subclass
techGetSymRules()	Returns symbolic rules defined in the symRules subclass

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:

SKILL Function	Description
techSetPrRoutingLayers	Overwrites the existing prRoutingLayers subclass with the specified data
techSetPrRoutingLayer	Appends or updates the specified layer and direction in the prRoutingLayers subclass
techGetPrRoutingLayers	Returns the layers listed in the prRoutingLayers subclass
techGetPrRoutingDirection	Returns the routing direction for the specified layer
techIsPrRoutingLayer	Returns t if the specified layer is listed in the prRoutingLayers subclass
techSetPrViaTypes	Overwrites the existing prViaTypes subclass with the specified data
techSetPrViaType	Appends or updates the specified via type in the prViaTypes subclass
techGetPrViaTypes	Returns the via and via type listed in the prViaTypes subclass
techGetPrViaType	Returns the via type of the specified device
techIsPrViaDevice	Returns t if the specified device is listed in the prViaType subclass
techSetPrStackVias	Overwrites the existing prStackVias subclass with the specified data

SKILL Function	Description
techSetPrStackVia	Appends or updates the specified layer in the prStackVias subclass
techGetPrStackVias	Returns the layers listed in the prStackVias subclass
techIsPrStackVia	Returns t if the specified device is listed in the prStackVias subclass
techSetPrMastersliceLayers	Overwrites the existing prMastersliceLayers subclass with the specified data
techSetPrMastersliceLayer	Appends or updates the specified layer to the prMastersliceLayers subclass
techGetPrMastersliceLayers	Returns the layers listed in the prMastersliceLayers subclass
techIsPrMastersliceLayer	Returns t if the specified device is listed in the prMastersliceLayers subclass
techSetPrViaRule	Updates the parameters for the specified via rule in the prViaRules subclass
techGetPrViaRules	Returns the via rules defined in the prViaRules subclass
techGetPrViaParams	Returns the parameters of the specified via rule defined in the prViaRules subclass
techSetPrGenViaRule	Updates the parameters for the specified generated via rule in the prGenViaRules subclass
techGetPrGenViaRules	Returns the generated via rules defined in the prGenViaRules subclass
techGetPrGenViaParams	Returns the parameters of the specified generated via rule defined in the prGenViaRules subclass
techSetPrNonDefaultRule	Updates the parameters for the specified nondefault rule in the prNonDefaultRules subclass
techGetPrNonDefaultRules	Returns the nondefault rules from the prNonDefaultRules subclass

SKILL Function	Description
techGetPrNonDefaultParams	Returns the parameters of the specified nondefault rule from the prNonDefaultRules subclass
techSetPrRoutingPitch	Appends or updates the specified layer in the prRoutingPitch subclass
techGetPrRoutingPitch	Returns the layers listed in the prRoutingPitch subclass
techSetPrRoutingOffset	Appends or updates the specified layer in the prRoutingOffset subclass
techGetPrRoutingOffset	Returns the layers listed in the prRoutingOffset subclass
techSetPrOverlapLayer	Appends the specified layer to the prOverlapLayer subclass
techSetPrOverlapLayers	Updates the specified layer in the prOverlapLayer subclass
techGetPrOverlapLayers	Returns the layers listed in the prOverlaplayer subclass

Checking a Technology File for Conformance to Cadence Application Requirements

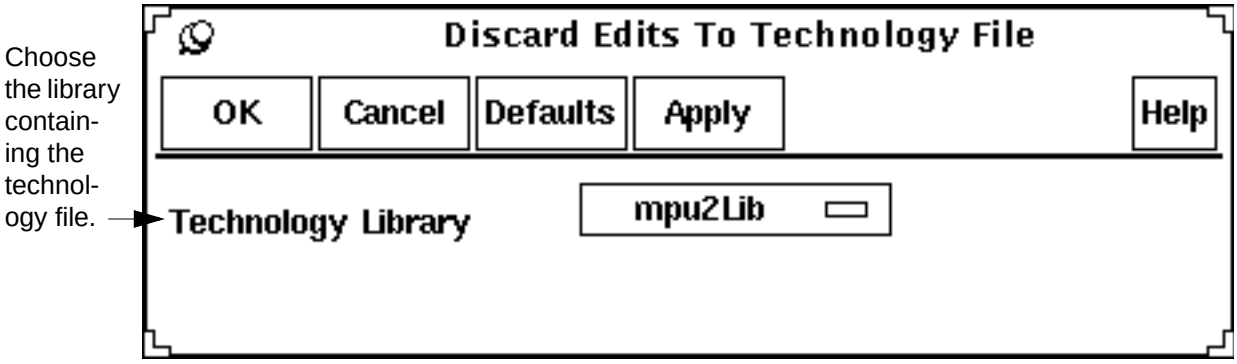
After you have compiled your ASCII technology file, you can check your file for use with the Cadence design applications. See "Checking a Technology File for Conformance to Application Requirements" on page 171 for details.

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 Technology File Tool Box, choose *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 DFII CIW and in the `techManager.log` file indicates that the command was successful.

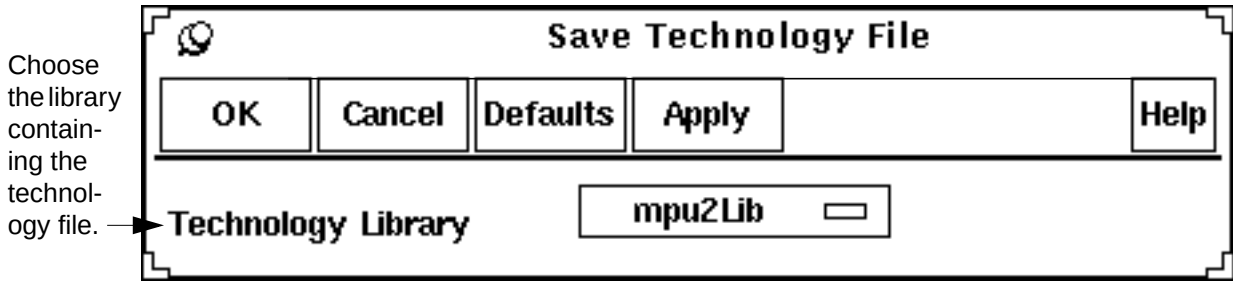
   ```
   Technology file 'cellTechLib' was restored successfully.
   ```

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 Technology File Tool Box, choose *Save*.

   The Save 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 you want to save.

3. Click *OK*.

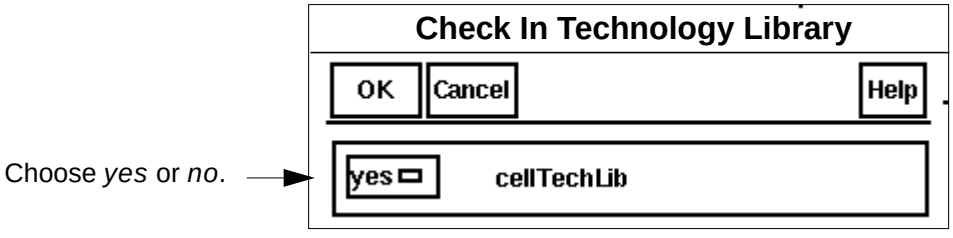
   A dialog box appears, asking you to confirm the save to disk. Click *Yes*.
   The technology file in virtual memory is saved to disk.

   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 with DFII, see "Setting Automatic Checkout and Checkin Preferences" in the *Design Framework II User Guide*.

4. If you are prompted to check in the technology file, choose a check-in option and click *OK*.



   If you choose *yes*, your changes are saved to disk and checked in.

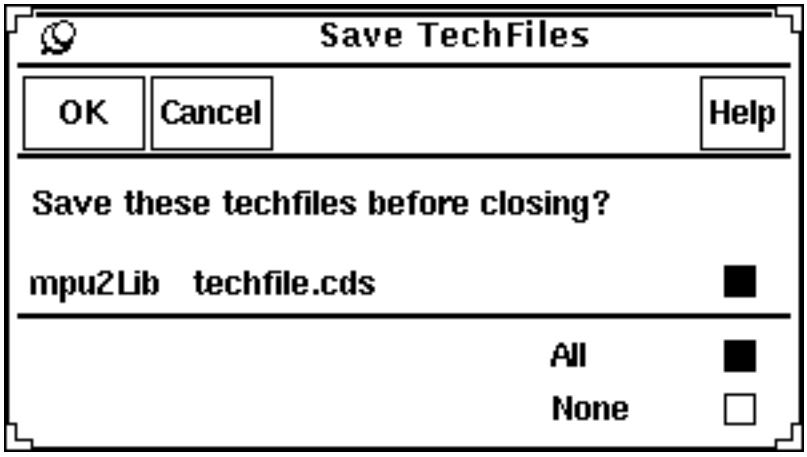
If you choose *no*, your changes are saved to disk but are not checked in.

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

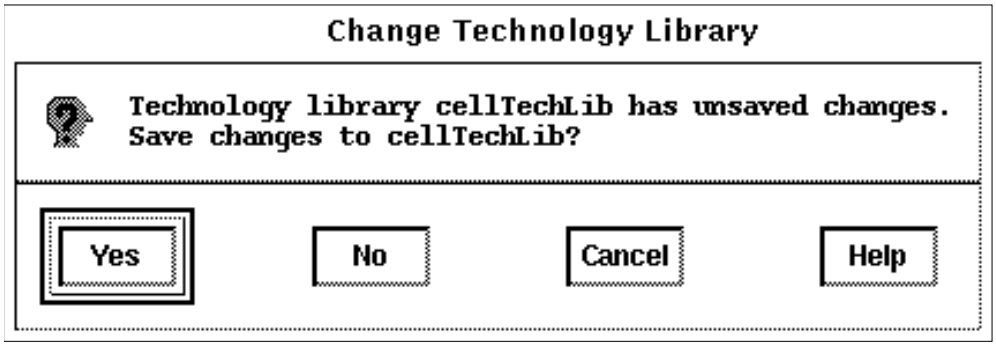
If you try to quit the software before saving your changes, you get a message box like the following example:



➤ To save the changes, set the radio buttons and click *OK*.

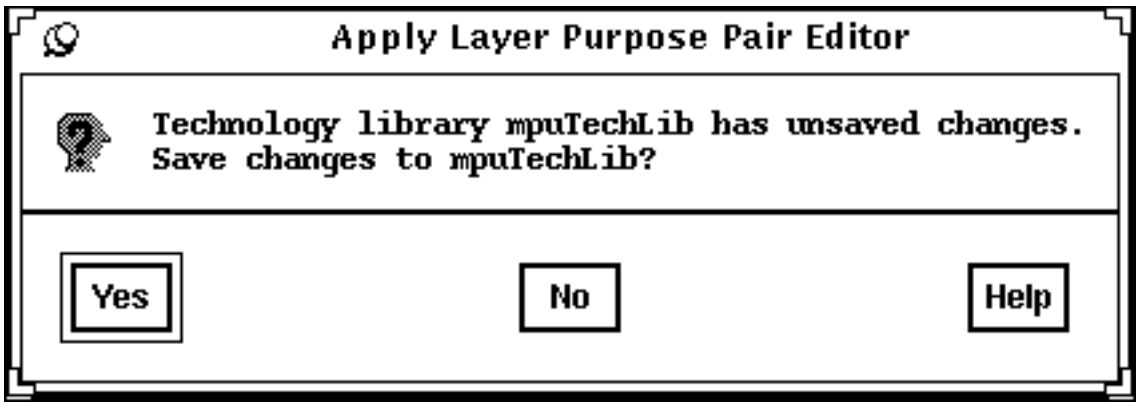
➤ To discard the changes, click *Cancel*.

If you try to change technology libraries before saving your changes, you get a message box like the following example:



- 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:



- To save the changes, click *Yes*.
- To discard the changes, click *No*.

9

Editing Class Data through the Technology File Tool Box: Controls and Layer Definitions

This chapter discusses the following:

Editing Controls Class Data

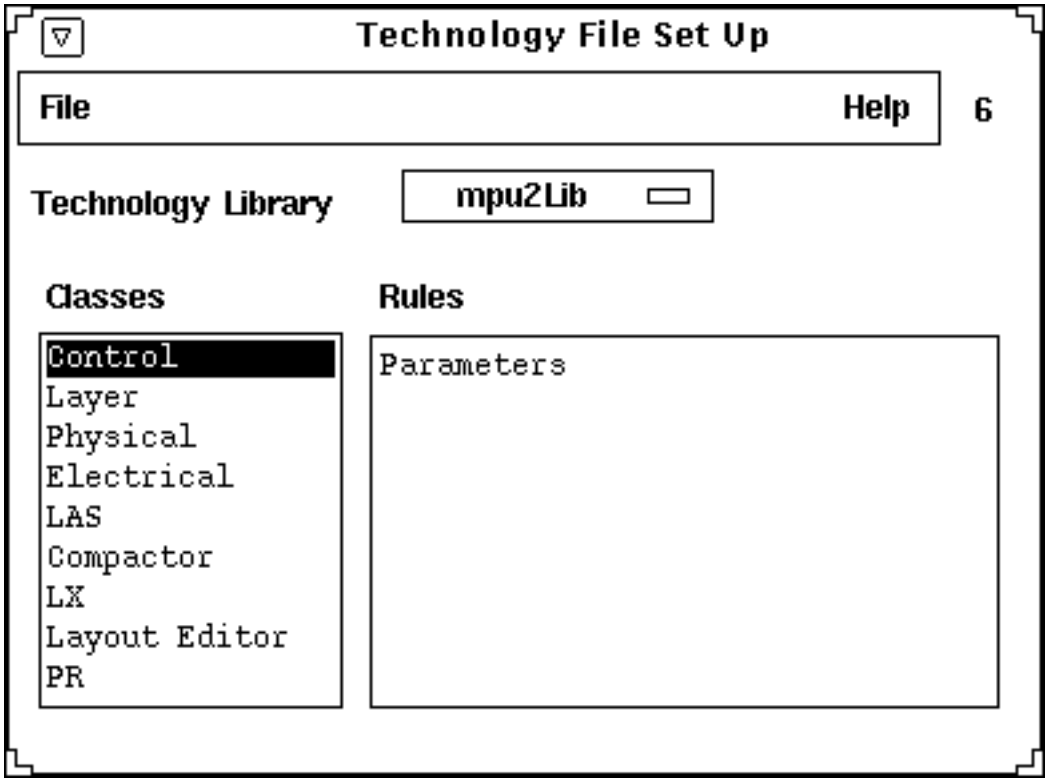
From the Technology File Tool Box, you can perform the following Controls class editing functions:

- 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 Technology File Tool Box, choose *Edit Rules*.

   The Technology File Set Up form appears.

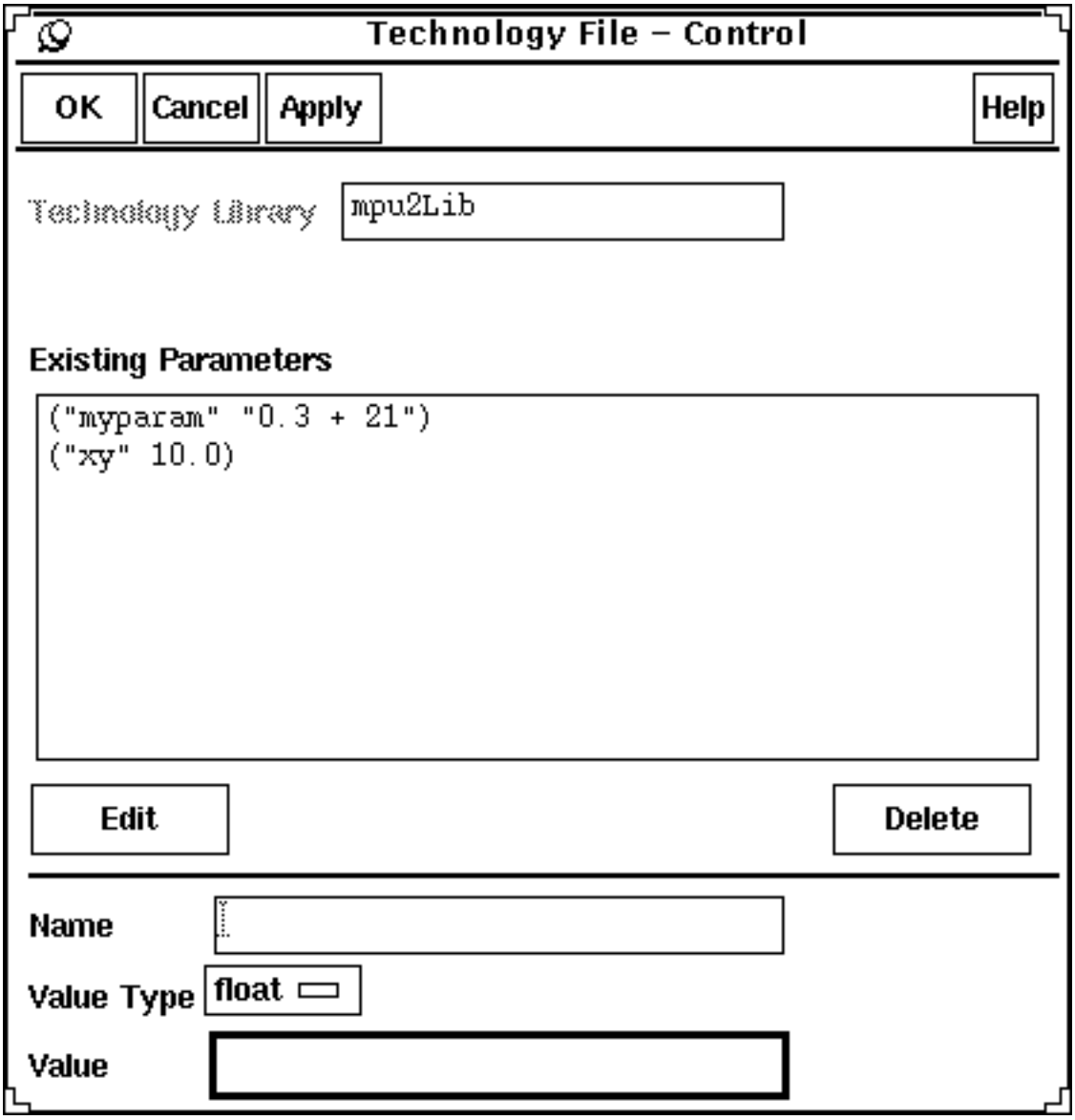


2. From the *Technology Library* cyclic field, choose the technology library containing the technology file to edit.
3. In the *Classes* list box, click on *Control* and, from the menu banner, choose *File – Edit*.

   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.



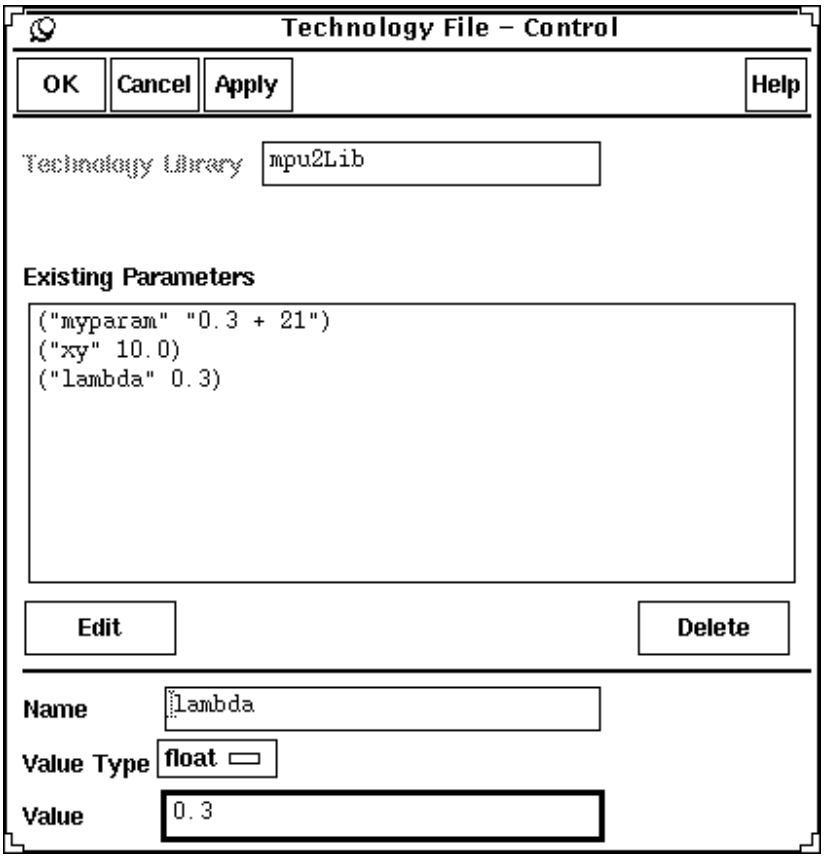
For a description of this form, see Appendix A.

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* 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 you want 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 *Save* from the Technology File Tool Box.

Editing Layer Definitions Class Data

From the Technology File Tool Box, you can perform the following Layer Definitions class editing functions:

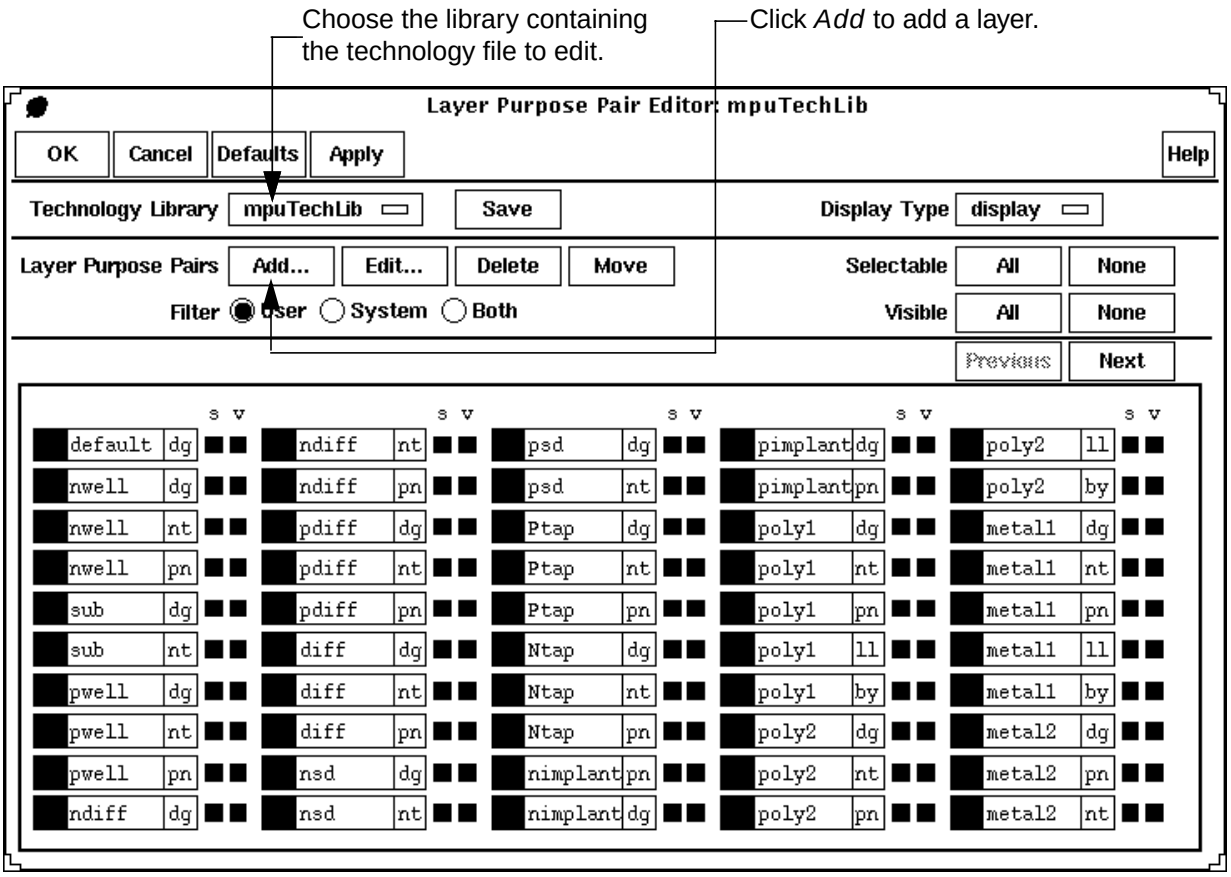
- Add a new layer definition (`layerDefinitions` class)
- Edit the display packet assigned to a layer (`layerDefinitions` class, `techDisplays` subclass)
- Edit layer display attributes and Stream rules (`layerDefinitions` class, `techDisplays` subclass; `layerRules` class, `streamLayers` subclass)
- 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 Technology File Tool Box, choose *Edit Layers*.

   The Layer Purpose Pair Editor form appears.



For a description of this form, see Appendix A.

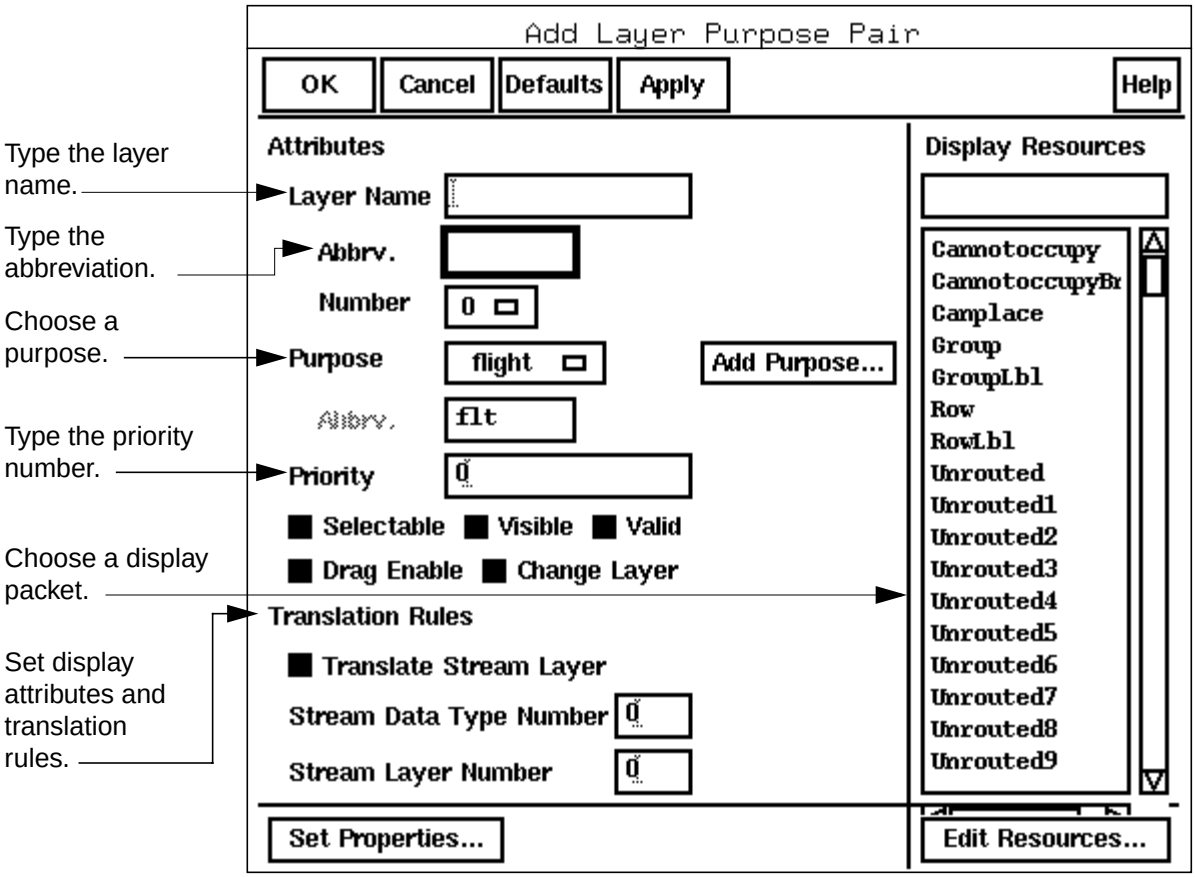
Design manager users click here for more information.

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 *Abbrv.* 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 *Abbrv.* 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.

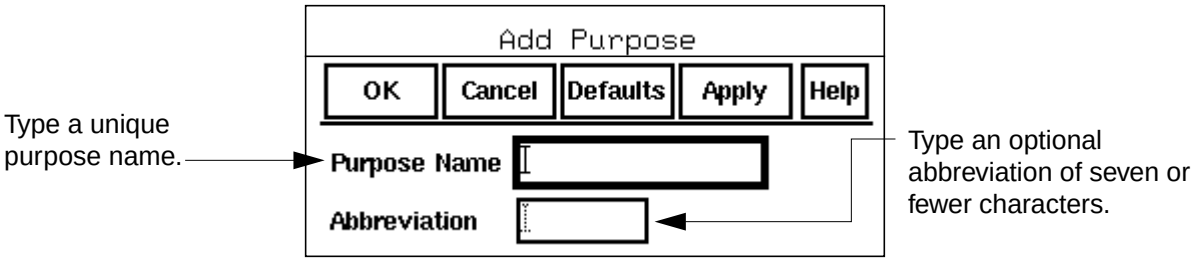
Note: The layer's number uniquely identifies the layer and distinguishes it from other layers; the number does not affect the layer's display priority.

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:

- ❑ 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 312.

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` subclass 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 subclass before it will appear in the LSW. For information about adding `leLswLayers` with the Technology File Tool Box, see "Editing Layout Editor Rules Class Data" on page 270.

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:

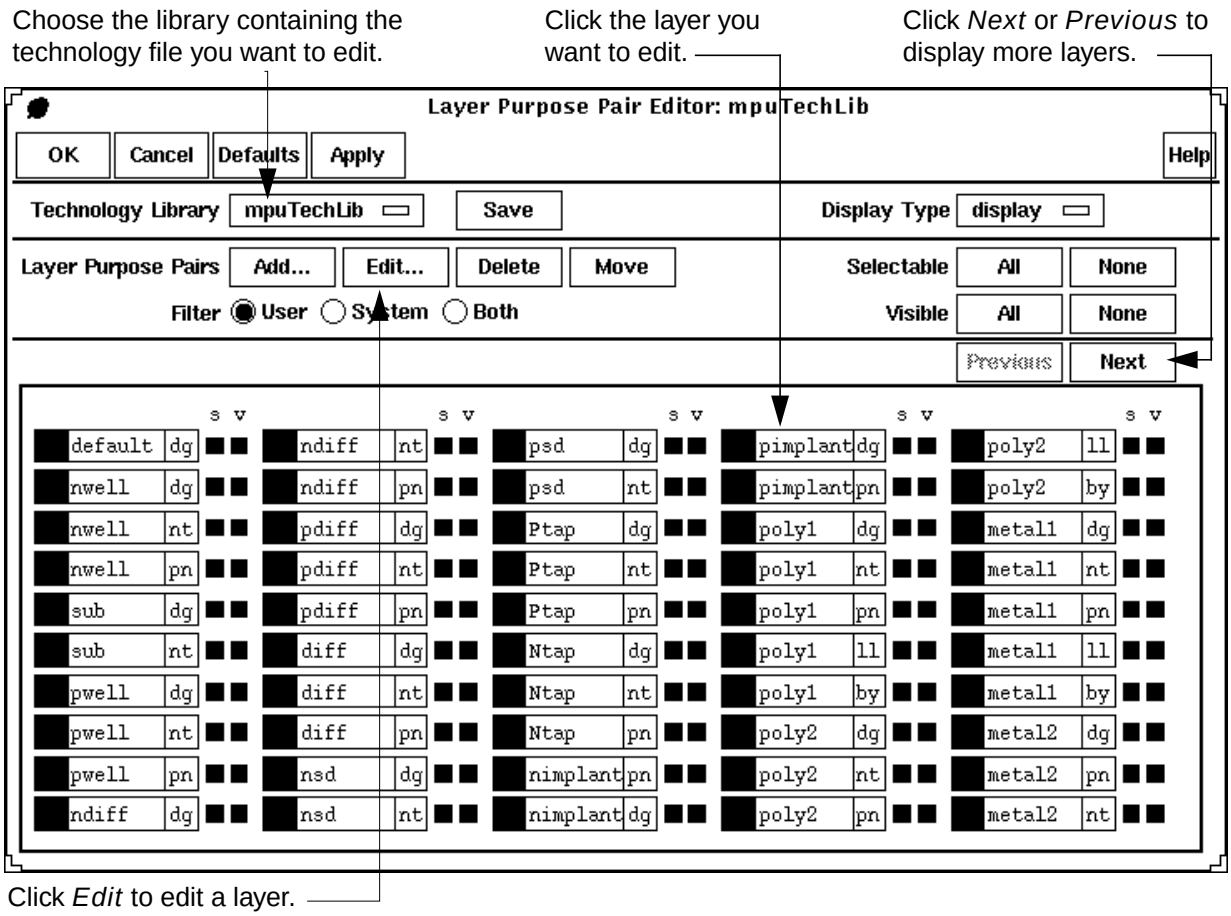
- Edit the display packet definition
- Assign a different display packet to the layer

For more information about editing display packets, refer to "Changing a Display Packet Definition" on page 312.

To assign a different display packet to the layer, do the following:

1. From the Technology File Tool Box, choose *Edit Layers*.

   The Layer Purpose Pair Editor 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.

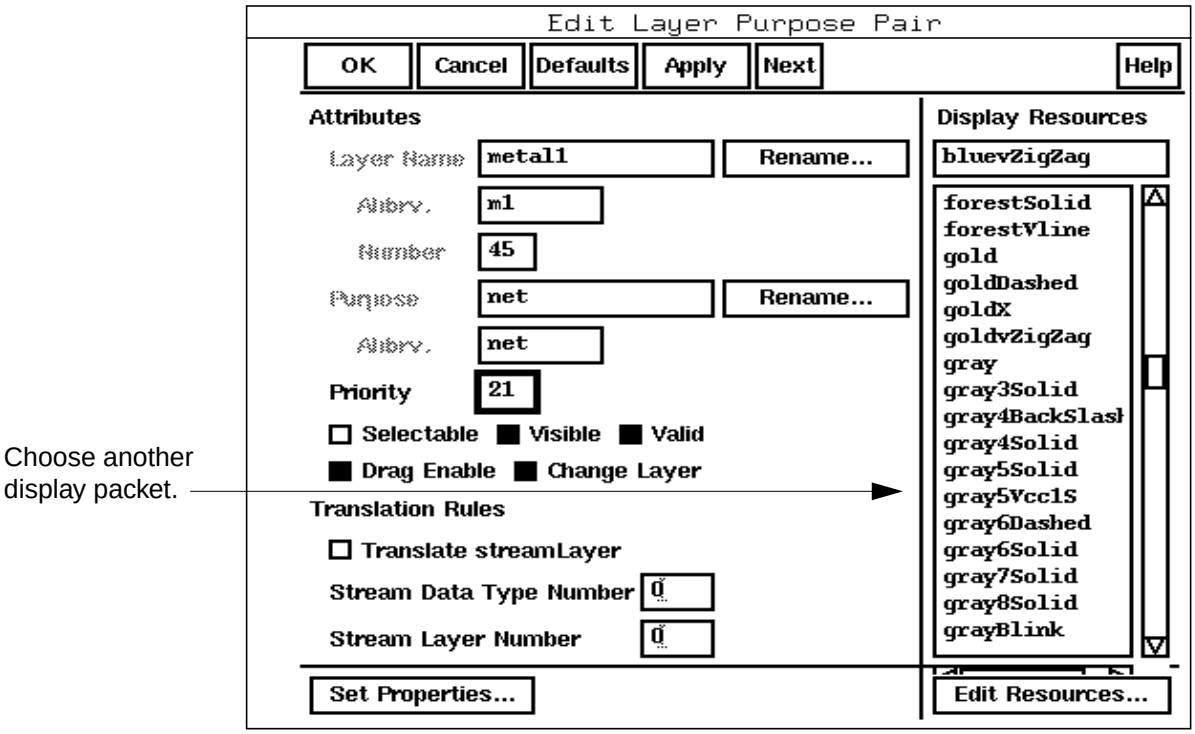
The form is updated with layers from the technology file you chose.

3. Click the layer you want to edit.

Note: The Layer Purpose Pair Editor displays a maximum of 50 layer-purpose pairs at a time. When there are more layer-purpose pairs, the *Next* button is enabled. Click *Next* to display the next page of layer-purpose pairs to choose from. The *Previous* button is enabled when you page forward through layer-purpose pairs. When the *Next* button is disabled, there are no further pages of layer-purpose pairs ahead of the currently displayed list; when the *Previous* button is disabled, there are no further pages of layer-purpose pairs behind the currently displayed list.

4. Click *Edit*.

   The Edit Layer Purpose Pair form appears.



   For a description of this form, see Appendix A.

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*.

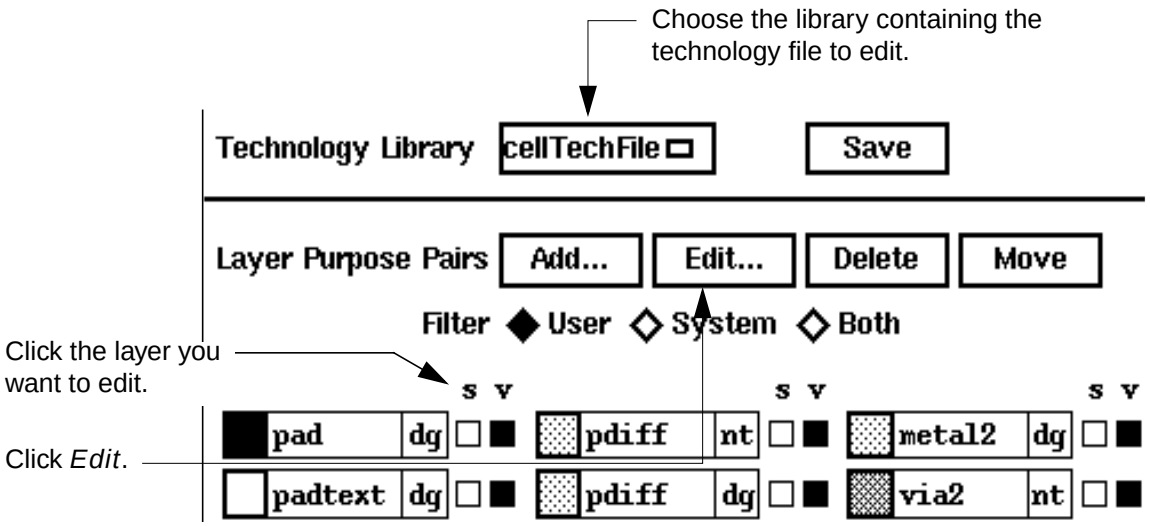
   Design manager users click here for more information.

Editing Layer Attributes

To edit the layer attributes, do the following:

1. From the Technology File Tool Box, choose *Edit Layers*.

   The Layer Purpose Pair Editor appears. The following shows the part of the form you use:



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.

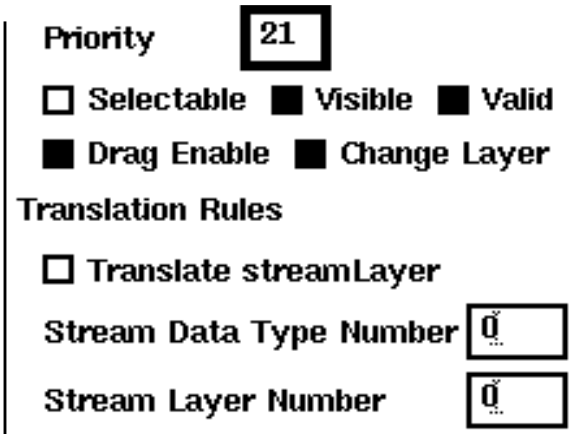
3. Click the layer you want to edit.

   Note: The Layer Purpose Pair Editor displays a maximum of 50 layer-purpose pairs at a time. When there are more layer-purpose pairs, the *Next* button is enabled. Click *Next* to display the next page of layer-purpose pairs to choose from. The *Previous* button is enabled when you page forward through layer-purpose pairs. When the *Next* button is disabled, there are no further pages of layer-purpose pairs ahead of the currently displayed list; when the *Previous* button is disabled, there are no further pages of layer-purpose pairs behind the currently displayed list.

4. Click *Edit*.

The Edit Layer Purpose Pair form appears. The following shows the part of the form you change:

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*.

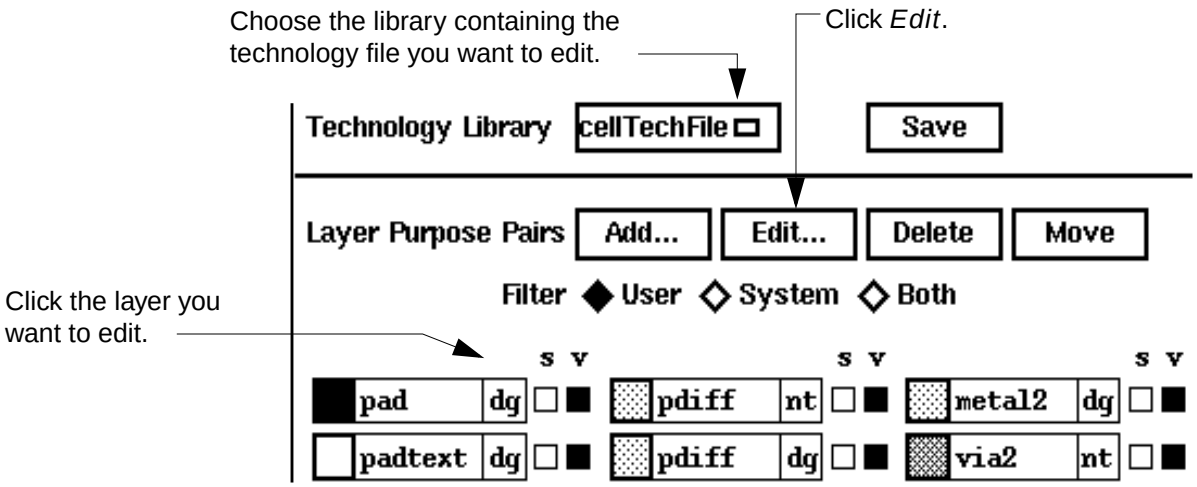
   Design manager users click here for more information.

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 Technology File Tool Box, choose *Edit Layers*.

The Layer Purpose Pair Editor form appears. The following shows the part of the Layer Purpose Pair Editor form you use:



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.

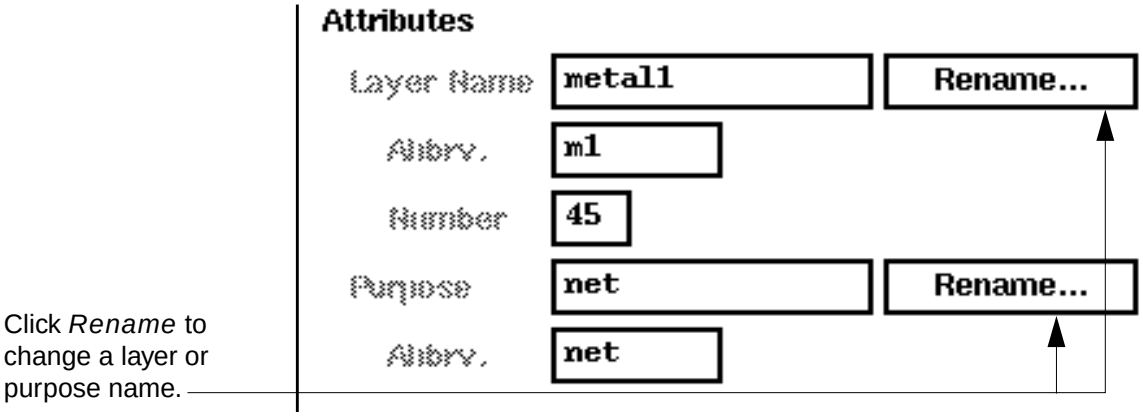
   The form is updated with layers from the technology file you chose.

3. Click the layer you want to edit.

   Note: The Layer Purpose Pair Editor displays a maximum of 50 layer-purpose pairs at a time. When there are more layer-purpose pairs, the *Next* button is enabled. Click *Next* to display the next page of layer-purpose pairs to choose from. The *Previous* button is enabled when you page forward through layer-purpose pairs. When the *Next* button is disabled, there are no further pages of layer-purpose pairs ahead of the currently displayed list; when the *Previous* button is disabled, there are no further pages of layer-purpose pairs behind the currently displayed list.

4. 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:

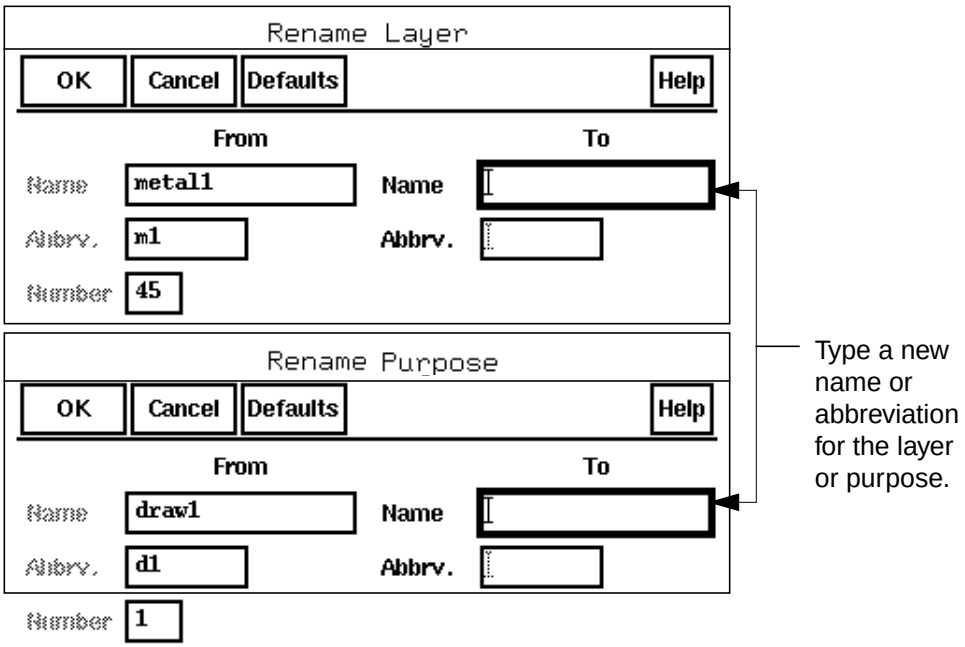


For a description of this form, see Appendix A.

5. 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.



For a description of these forms, see Appendix A.

6. 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.

7. 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).

8. To close the Edit Layer Purpose Pair form, click *OK*.

9. To close the Layer Purpose Pair Editor form, click *OK*.

   Design manager users click here for more information.

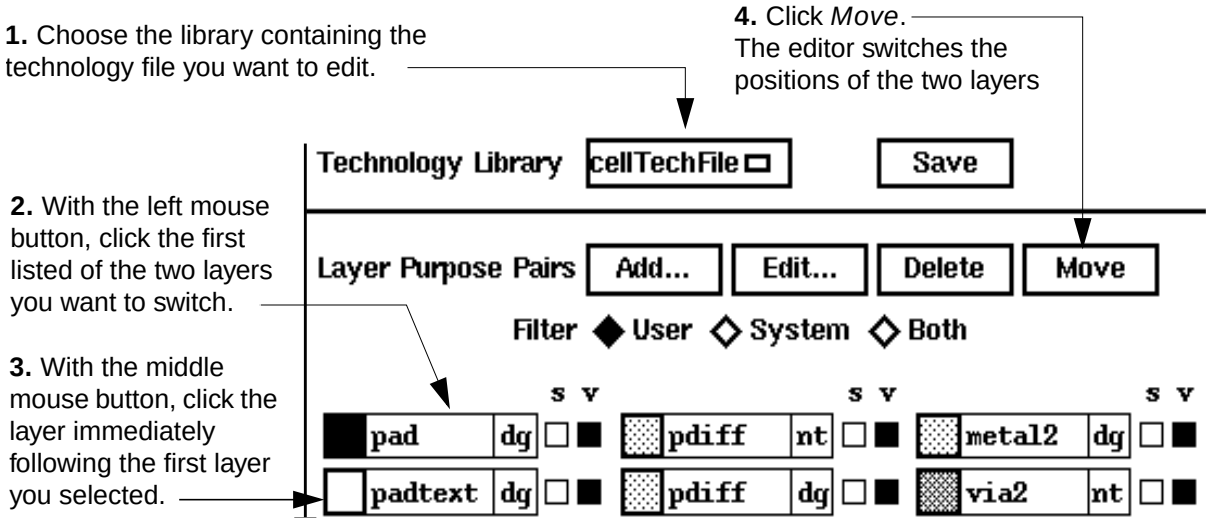
Changing Layer Priorities

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 Technology File Tool Box, choose *Edit Layers*.

   The Layer Purpose Pair Editor appears. The following shows the part of the Layer Purpose Pair Editor form you use:



   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.

   Note: The Layer Purpose Pair Editor displays a maximum of 50 layer-purpose pairs at a time. When there are more layer-purpose pairs, the *Next* button is enabled. Click *Next* to display the next page of layer-purpose pairs to choose from. The *Previous* button is enabled when you page forward through layer-purpose pairs. When the *Next* button is disabled, there are no further pages of layer-purpose pairs ahead of the currently displayed list; when the *Previous* button is disabled, there are no further pages of layer-purpose pairs behind the currently displayed list.

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*.

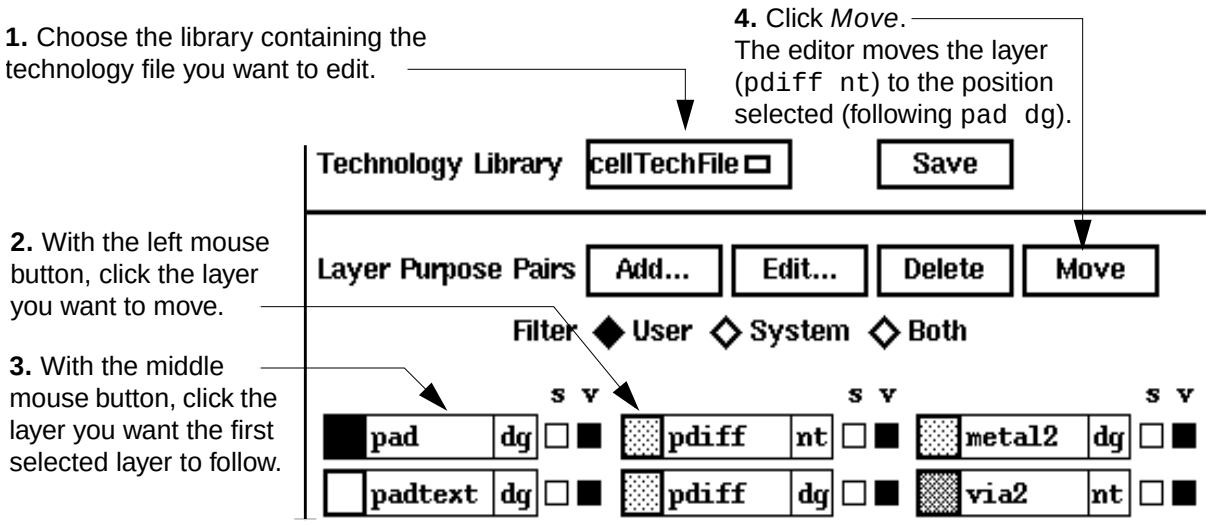
   Design manager users click here for more information.

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 Technology File Tool Box, choose *Edit Layers*.

The Layer Purpose Pair Editor appears. The following shows the part of the Layer Purpose Pair Editor form you use:



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.

   Note: The Layer Purpose Pair Editor displays a maximum of 50 layer-purpose pairs at a time. When there are more layer-purpose pairs, the *Next* button is enabled. Click *Next* to display the next page of layer-purpose pairs to choose from. The *Previous* button is enabled when you page forward through layer-purpose pairs. When the *Next* button is disabled, there are no further pages of layer-purpose pairs ahead of the currently displayed list; when the *Previous* button is disabled, there are no further pages of layer-purpose pairs behind the currently displayed list.

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.

   Note: The second layer-purpose pair can be on a different display page reached with the *Next* or *Previous* 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*.

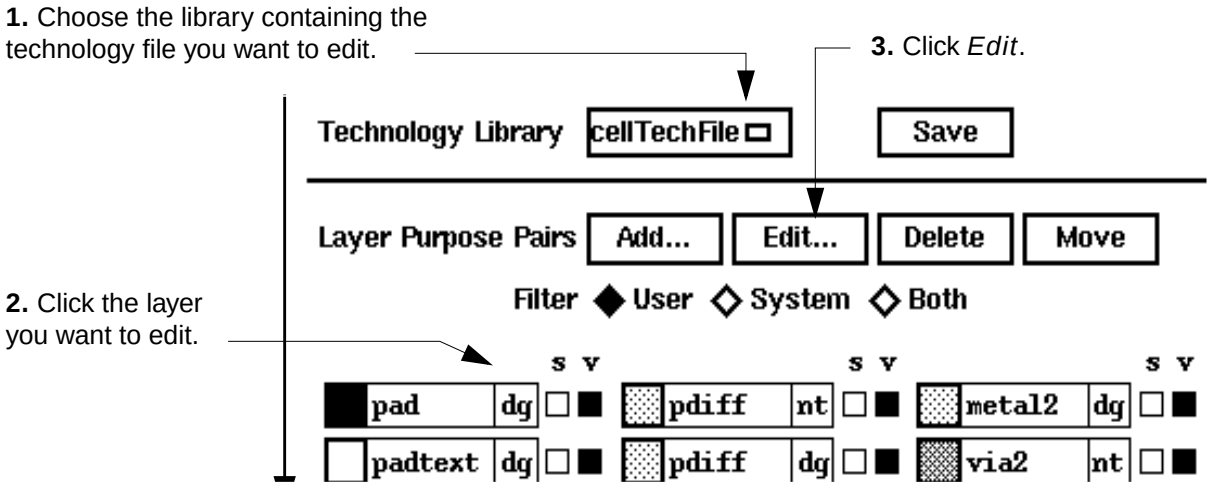
   Design manager users click here for more information.

Adding or Editing Layer Properties

Cadence® design 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 Technology File Tool Box, choose *Edit Layers*.

   The Layer Purpose Pair Editor form appears. The following shows the part of the Layer Purpose Pair Editor form you use:



   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.

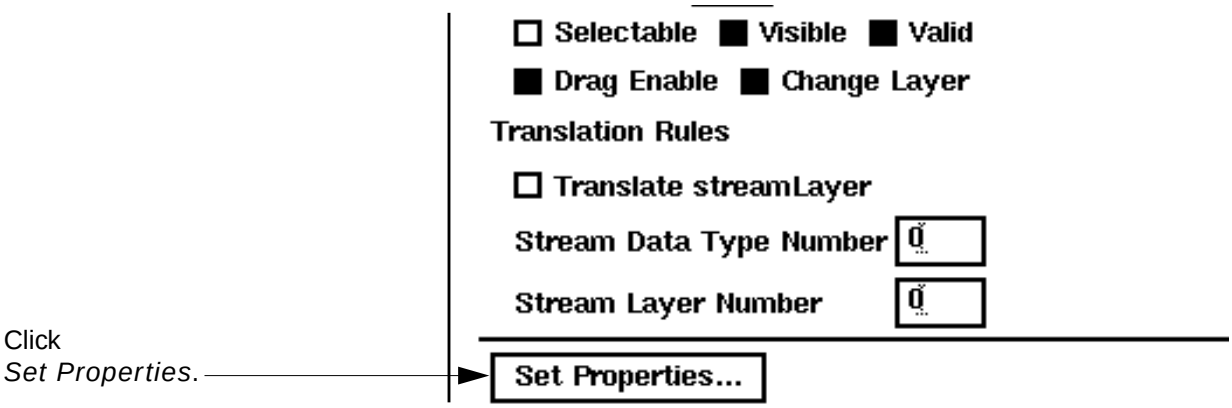
3. Click the layer you want to edit.

Note: The Layer Purpose Pair Editor displays a maximum of 50 layer-purpose pairs at a time. When there are more layer-purpose pairs, the *Next* button is enabled. Click *Next* to display the next page of layer-purpose pairs to choose from. The *Previous* button is enabled when you page forward through layer-purpose pairs. When the *Next* button is disabled, there are no further pages of layer-purpose pairs ahead of the currently displayed list; when the *Previous* button is disabled, there are no further pages of layer-purpose pairs behind the currently displayed list.

4. Click *Edit*.

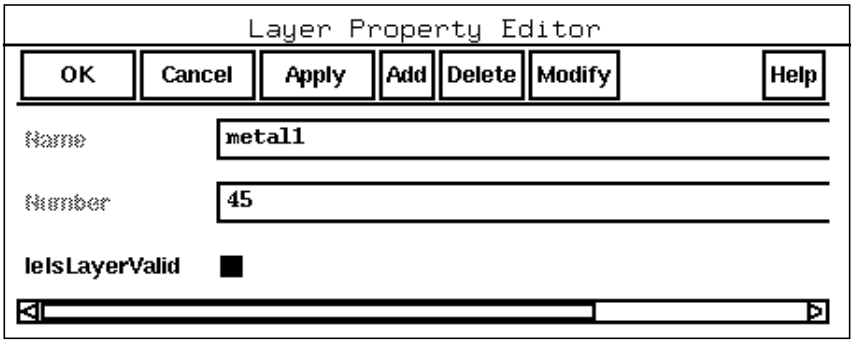
The Edit Layer Purpose Pair form appears.

5. To add or edit properties for the layer, click *Set Properties*.



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.



For a description of this form, see Appendix A.

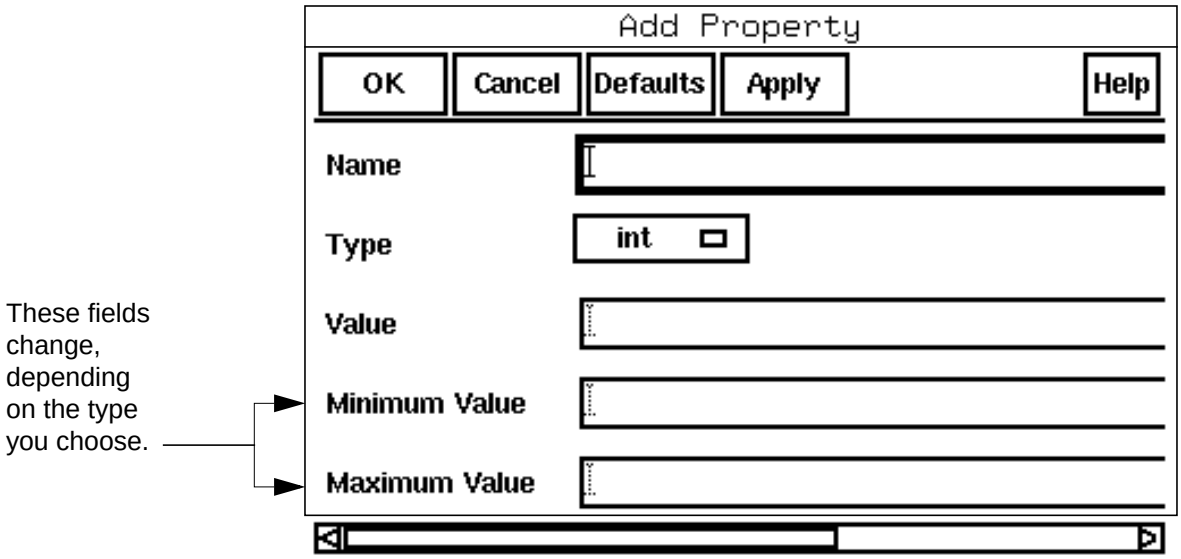
6. To delete a property, choose the property name and click *Delete*.

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.



For a description of this form, see Appendix A.

- ❑ Type a name, choose a type, and type a value.
- ❑ Click *OK*.

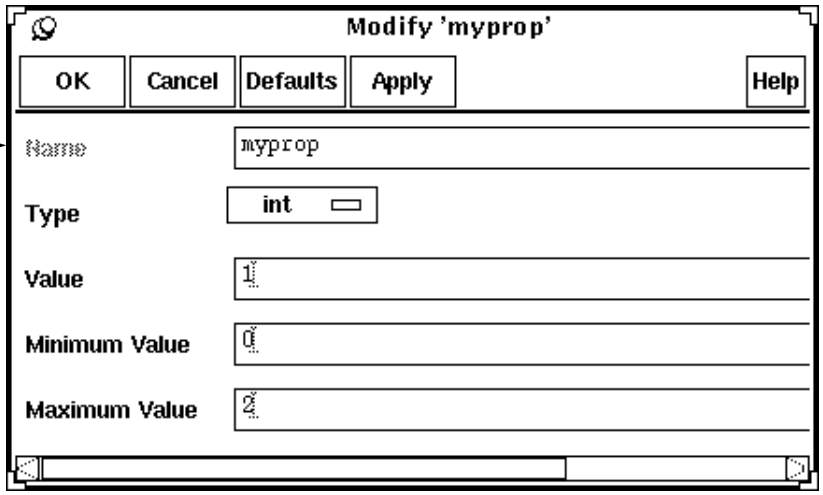
The property is added to the form and to the technology file in virtual memory.

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.

This form is the same as the Add Property form, except you cannot change the property name.



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.

Note: The Layer Purpose Pair Editor displays a maximum of 50 layer-purpose pairs at a time. When there are more layer-purpose pairs, the *Next* button is enabled. Click *Next* to display the next page of layer-purpose pairs to choose from. The *Previous* button is enabled when you page forward through layer-purpose pairs. When the *Next* button is disabled, there are no further pages of layer-purpose pairs ahead of the currently displayed list; when the *Previous* button is disabled, there are no further pages of layer-purpose pairs behind the currently displayed list. The layers you select can be on different display pages reached with the *Next* or *Previous* 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 following 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

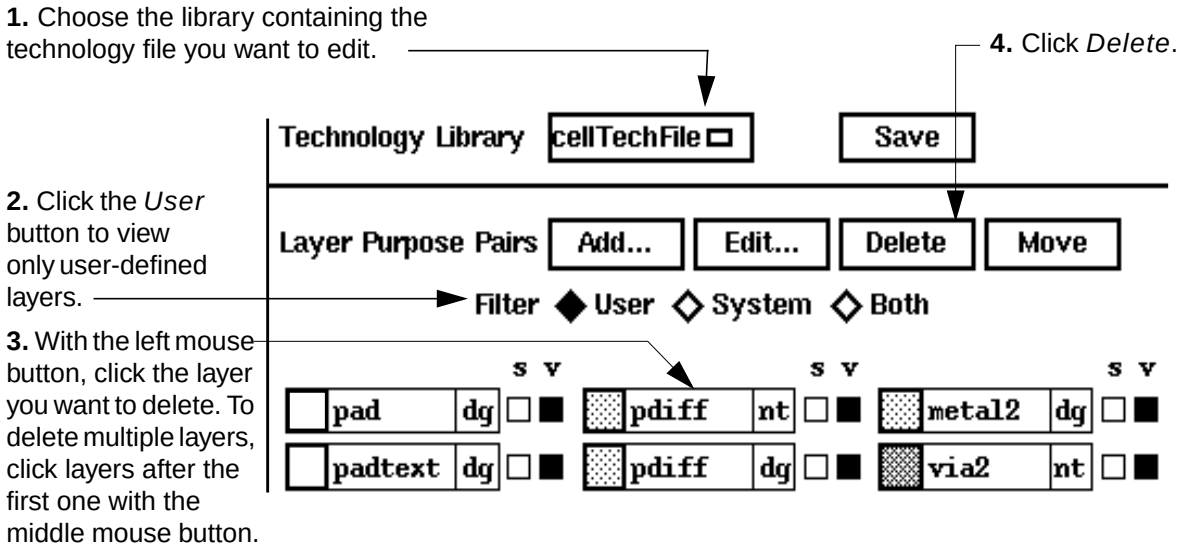
The layer is the most critical element in a technology file. All technology file classes can reference layers to create devices and specify how objects on different layers relate to each other. Do not delete a layer until you have removed all references to it from the technology file.

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 Technology File Tool Box, choose *Edit Layers*.

The Layer Purpose Pair Editor form appears. The following shows the part of the Layer Purpose Pair Editor form you use:



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.

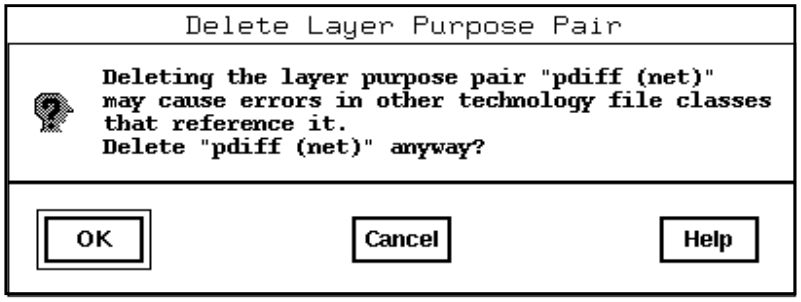
   You cannot delete a system-defined layer.

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.

   Note: The Layer Purpose Pair Editor displays a maximum of 50 layer-purpose pairs at a time. When there are more layer-purpose pairs, the *Next* button is enabled. Click *Next* to display the next page of layer-purpose pairs to choose from. The *Previous* button is enabled when you page forward through layer-purpose pairs. When the *Next* button is disabled, there are no further pages of layer-purpose pairs ahead of the currently displayed list; when the *Previous* button is disabled, there are no further pages of layer-purpose pairs behind the currently displayed list. You can select multiple layers on different display pages reached with the *Next* or *Previous* button.

5. Click *Delete*.

A dialog box appears, such as the following:



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.

10

Editing Class Data through the Technology File Tool Box: Generic Rules

This chapter discusses the following:

Editing Layer Rules Class Data

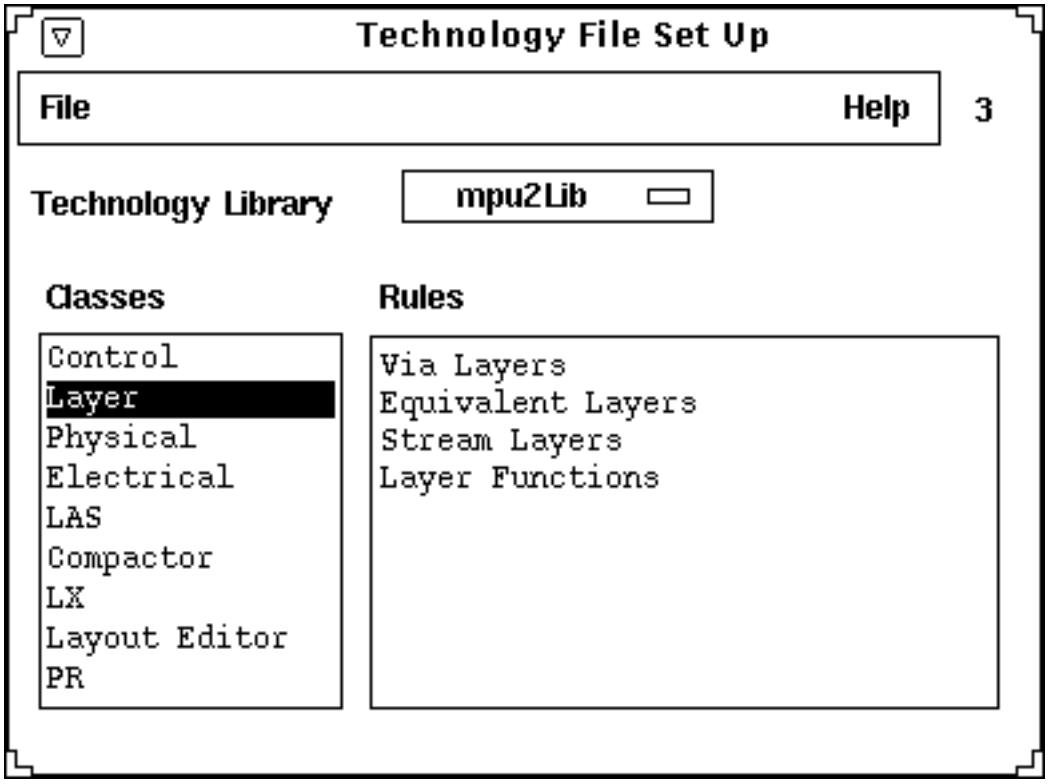
From the Technology File Tool Box, you can perform the following Layer Rules class editing functions:

- 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
- Edit or add layer function assignments

To edit Layer Rules class technology file data in virtual memory, perform the following steps:

1. From the Technology File Tool Box, choose *Edit Rules*.

   The Technology File Set Up form appears.



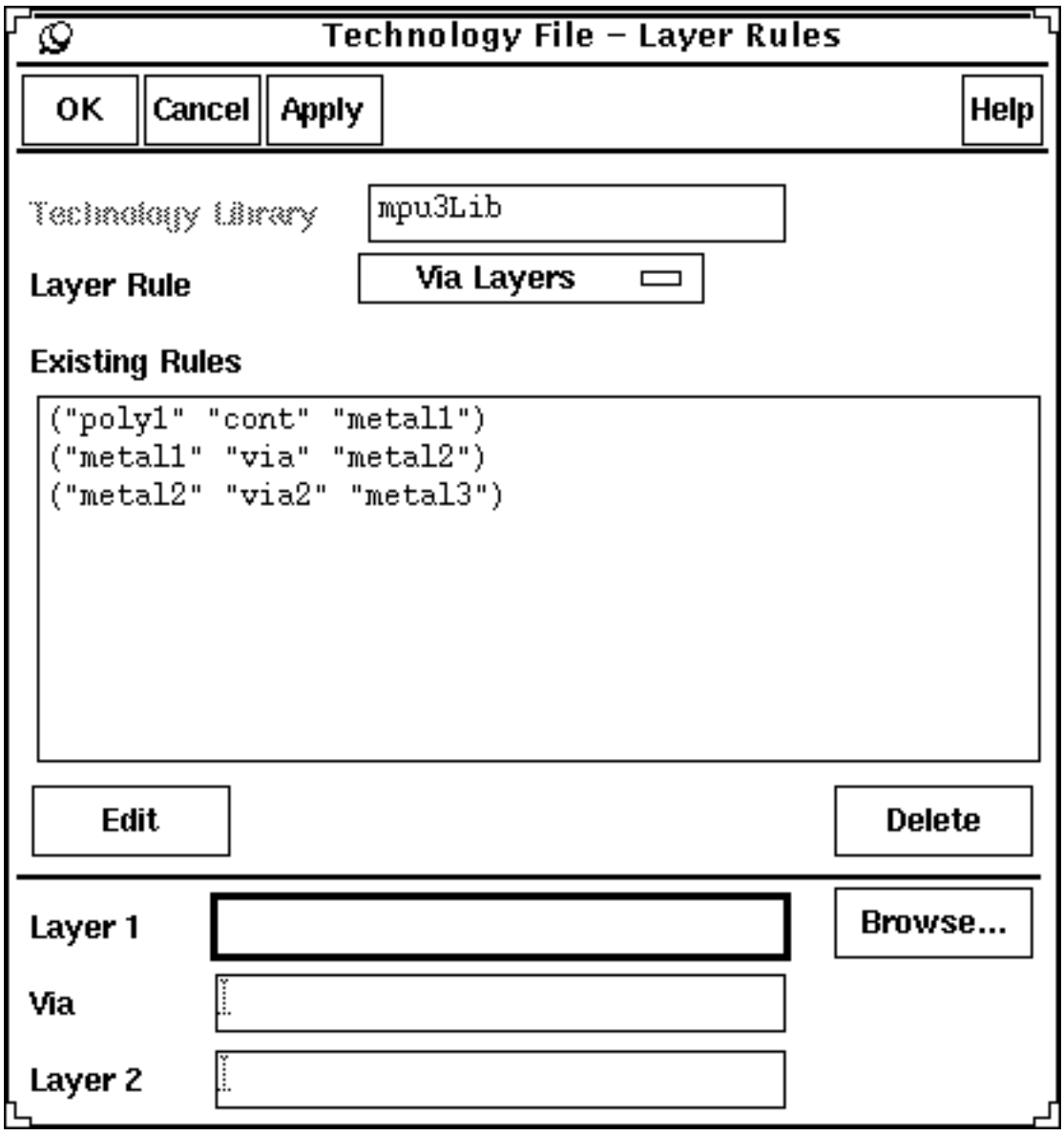
2. From the *Technology Library* cyclic field, choose the technology library containing the technology file to edit.

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.



For a description of this form, see Appendix A.

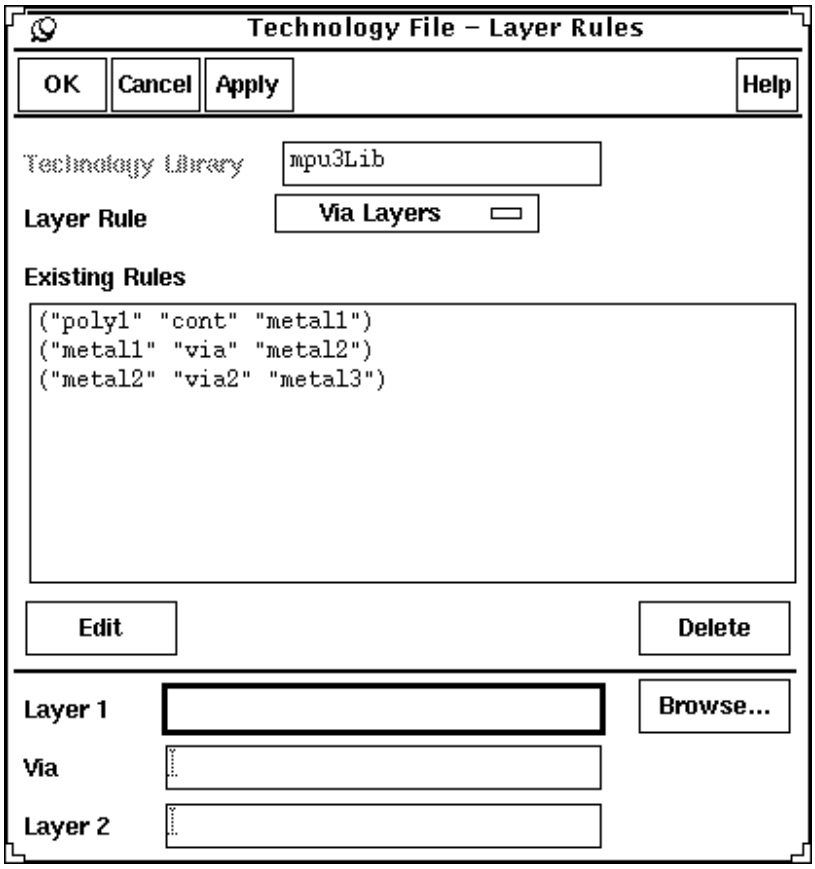
4. From the *Layer Rule* cyclic field, choose the subclass to edit from the following:

   - Via Layers
   - Equivalent Layers
   - Stream Layers
   - layerFunctions

   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

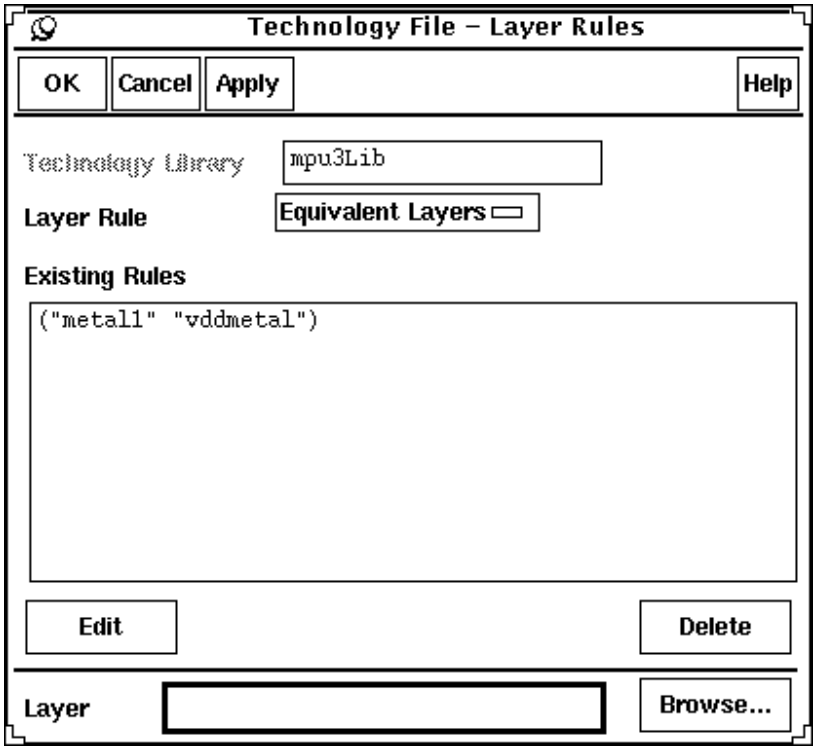
1. In the *Layer 1*, *Via*, and *Layer 2* fields at the bottom of the form, type the layer names. (You can click *Browse* to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the appropriate *Layer* 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 you want to delete.
2. Click *Delete*. The software removes the rule from the *Existing Rules* list box.

❑ equivalentLayers



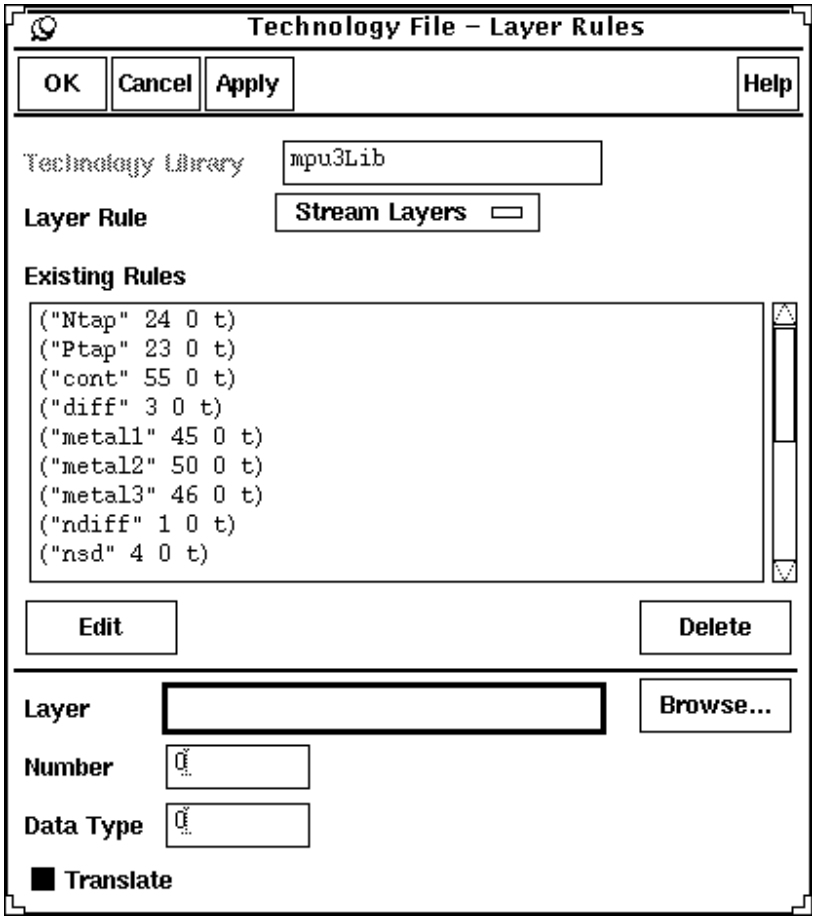
To add an `equivalentLayers` rule, use either of the following methods:

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 you want 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.

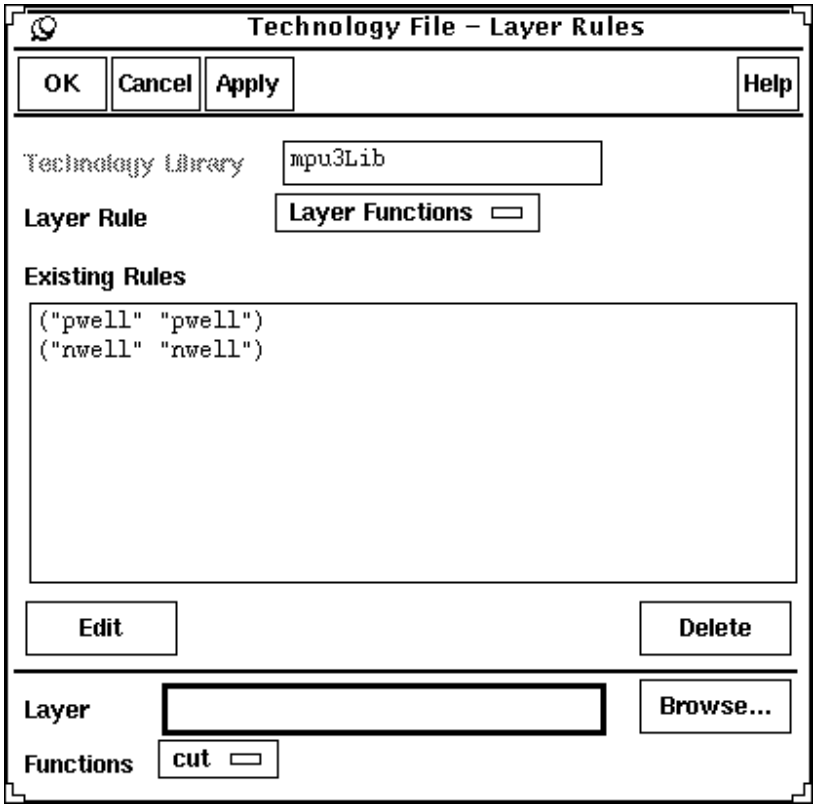
Method 2

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 you want 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.

- `layerFunctions`



To add a `layerFunctions` 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. (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 *Functions* cyclic field, choose the function to apply to the layer.
3. Click *Edit*. The software adds the new layer function definition 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 `layerFunctions` arguments in the *Layer* and *Functions* fields.
2. Edit the data in the *Layer* and *Functions* fields. **Note:** You must specify a new layer to produce a new `layerFunctions` rule.(You can use the Layer Browser as described in *Method 1*.)
3. Choose the appropriate *Functions* selection.
4. Click *Edit*. The software adds the new rule to the *Existing Rules* list box.

To delete a `layerFunctions` rule:
1. In the *Existing Rules* list box, click on the rule you want to delete.
2. Click *Delete*. The software removes the rule from the *Existing Rules* list box.

To edit an existing `layerFunctions` rule:
1. In the *Existing Rules* list box, click on the rule to edit. The form displays the `layerFunctions` arguments in the *Layer* and *Functions* fields.
2. From the *Functions* cyclic field, choose a new function to assign to the layer.
3. 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 *Save* from the Technology File Tool Box.

Editing Physical Rules Class Data

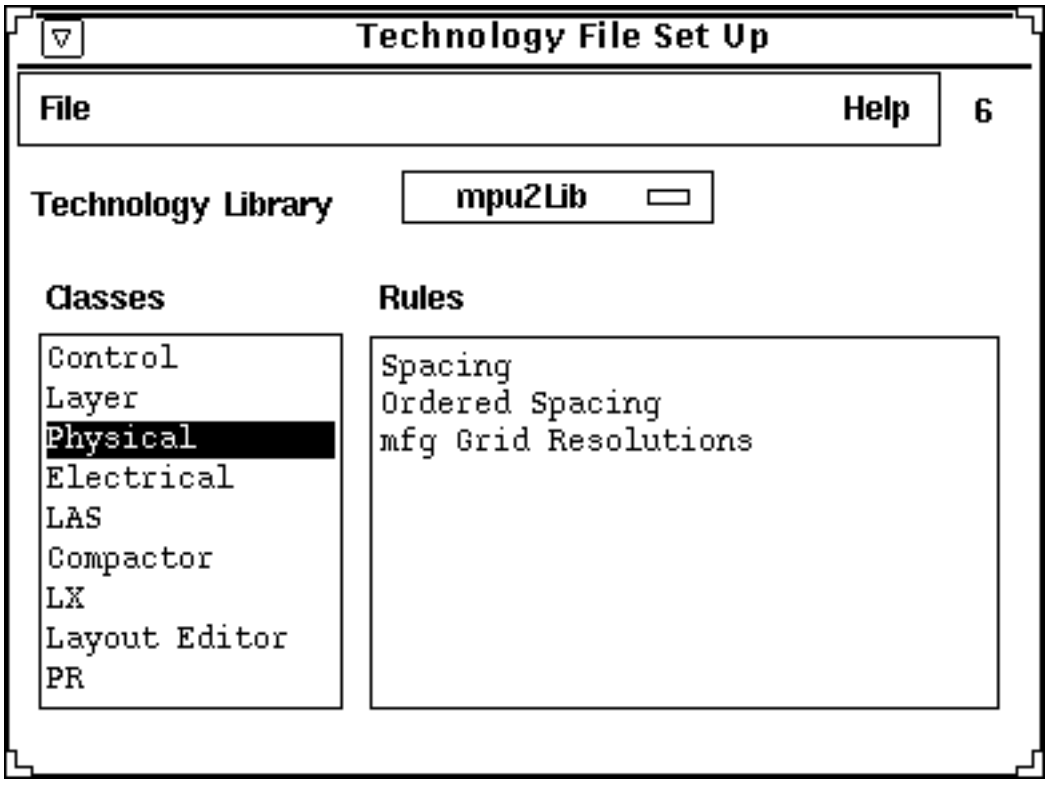
From the Technology File Tool Box, you can perform the following Physical Rules class editing functions:

- Add a `spacingRules` or `orderedSpacingRules` rule
- Delete a `spacingRules` or `orderedSpacingRules` rule
- Edit the value for an existing `spacingRules` or `orderedSpacingRules` rule
- Edit or add a spacing rules table
- 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 Technology File Tool Box, choose *Edit Rules*.

   The Technology File Set Up form appears.



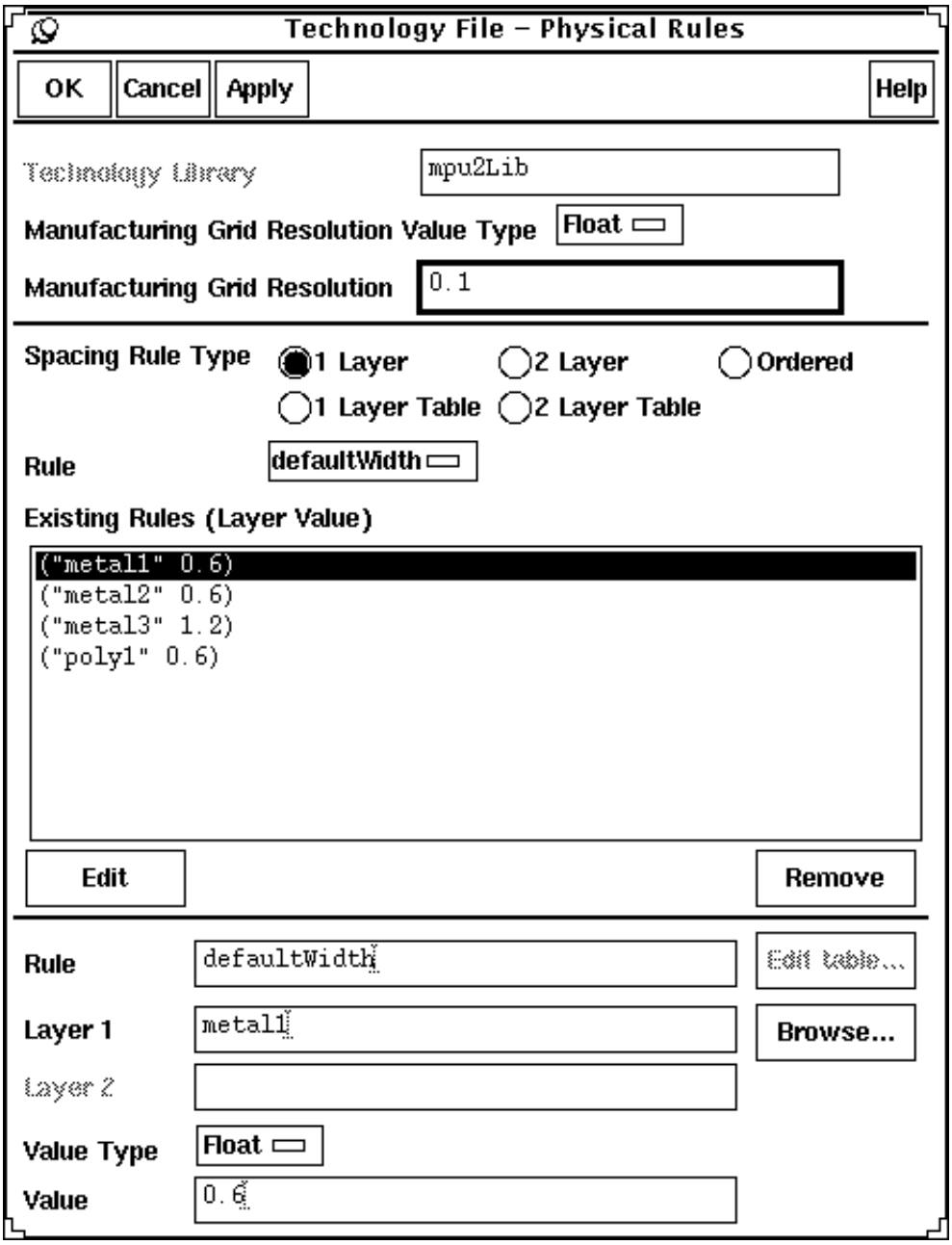
2. From the *Technology Library* cyclic field, choose the technology library containing the technology file to edit.

3. In the *Classes* list box, click on *Physical* and, from the form menu banner, choose *File – Edit*.

or

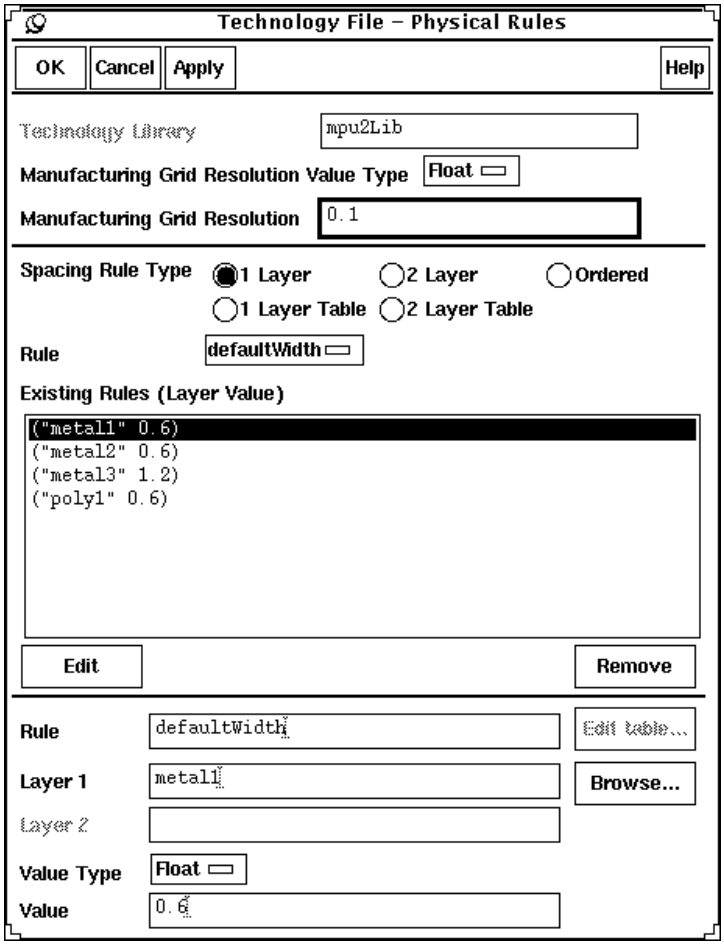
In the *Classes* list box, double-click on *Physical*.

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 `spacingRules` **or** `orderedSpacingRules` **rule**, use either of the following methods:



Method 1

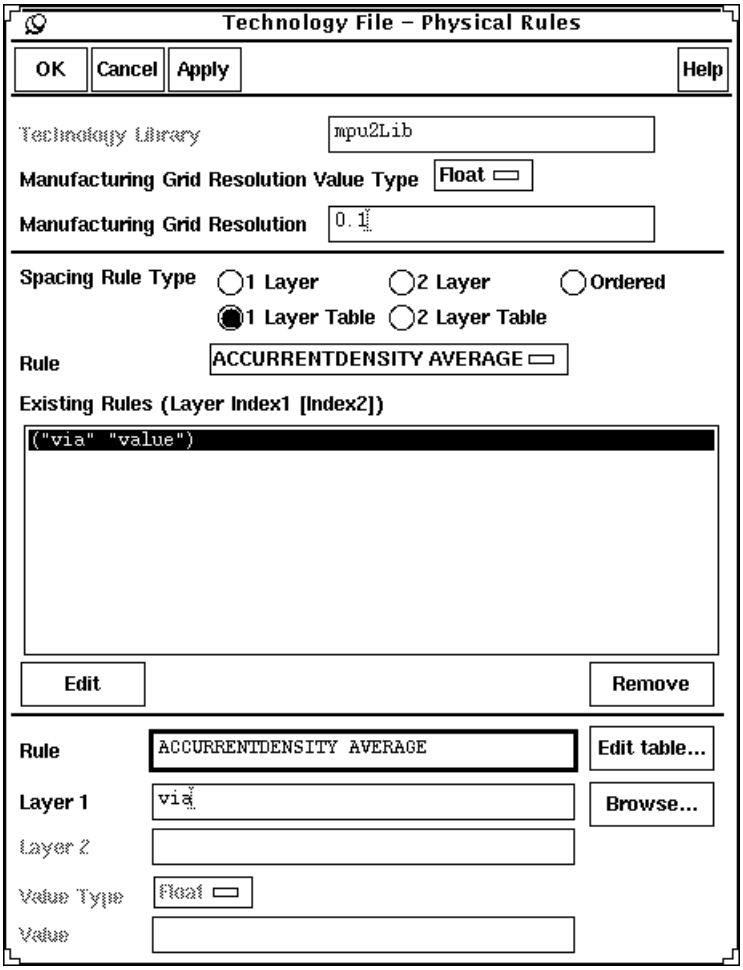
1. Click on the appropriate radio button to choose *1 Layer* to add a 1-layer spacing rule, *2 Layer* to add a 2-layer spacing rule, or *Ordered* to add an ordered-spacing rule.
2. In the fields at the bottom of the form, type the arguments for the new rule and choose the value type. (You can click *Browse* 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.)
3. Click *Edit*. The software adds the new spacing rule to the *Existing Rules* list box.

Method 2

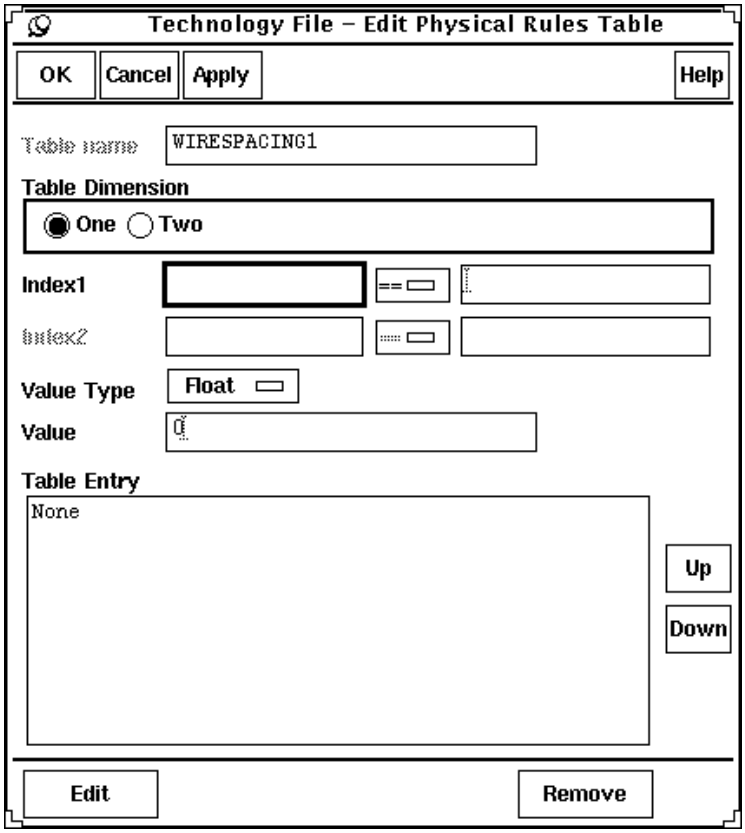
1. Click on the appropriate radio button to choose *1 Layer* or *2 Layer* to display `spacingRules` or to choose *Ordered* to display `orderedSpacingRules` in the *Existing Rules* list box.
2. From the *Rule* cyclic field, choose the rule name. The software displays the rules of the selected type with 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 add a `tableSpacingRules` **rule**, do the following:

1. Click on the appropriate radio button to choose *1 Layer Table* to add a 1-layer spacing rules table or *2 Layer Table* to add a 2-layer spacing rules table.
2. At the bottom of the form, in the *Rule* field, type the name of the rule table to create.
3. In the *Layer1* field, type the name of the layer for a 1-layer table or the first layer for a 2-layer table. For a 2-layer table, in the *Layer2* field, type the name of the second layer.
4. Click *Edit table*. The software displays the Technology File – Edit Physical Rules Table form.



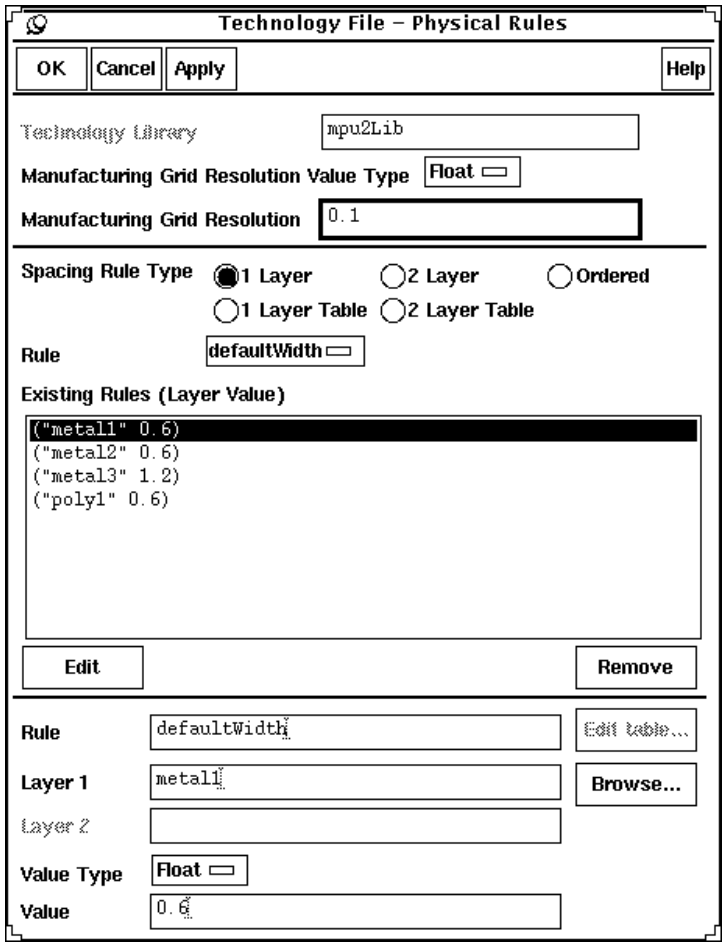
The form Technology File – Edit Physical Rules Table Form displays the rule name in the *Table name* field. For a description of this form, see Appendix A.



5. Click on the appropriate radio button to choose a one- or two-dimensional table.
6. In the first field of the *Index1* line, type the name of the first index. For a two-dimensional table, repeat for *Index2*.
7. From the match type cyclic field of the *Index1* line, choose the match type for the table entry index. For a two-dimensional table, repeat for *Index2*.
8. In the rightmost field of the *Index1* line, type the index to enter in the table entry. For a two-dimensional table, repeat for *Index2*.
9. From the *Value Type* cyclic field, choose the type for the value associated with the index or indices.
10. In the *Value* field, type the value to associate with the index or indices.
11. Click *Edit*. The software adds the table entry to the *Table Entry* list box.
12. Continue entering table entries until the table is complete.
13. When the table is complete, click *OK* to apply the table entries and dismiss the form.

To delete a table entry:
1. In the *Table Entry* list box, click on the table entry you want to delete.
2. Click *Remove*. The software removes the table entry from the *Table Entry* list box.

To delete a spacing rule:
1. Click on the appropriate radio button to choose *1 Layer* or *2 Layer* for `spacing-Rules`, to choose *Ordered* for `orderedSpacingRules`, or to choose *1 Layer Table* or *2 Layer Table* for `tableSpacingRules`.
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 you want to delete.
4. Click *Remove*. 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`, to choose *Ordered* for `orderedSpacingRules`, or to choose *1 Layer Table* or *2 Layer Table* for `tableSpacingRules`.
2. From the *Rule* cyclic field, choose the rule name. For `spacingRules` and `orderedSpacingRules`, the software displays the rules of the selected type with the selected rule name in the *Existing Rules* list box. For `tableSpacingRules`, the software displays the layer and index names for the table.
3. In the *Existing Rules* list box, click on the rule to edit. For `spacingRules` and `orderedSpacingRules`, the software displays the rule arguments in the data fields at the bottom of the form. For `tableSpacingRules`, the software displays the rule name and layer name or names.
4. For `spacingRules` and `orderedSpacingRules`, edit the data in the *Value Type* or *Value* fields. For `tableSpacingRules`, click *Edit table*, edit the table in the Edit Physical Rules Table form, and click *OK* when done.
5. On the Technology File – Physical Rules form, 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.

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 *Save* from the Technology File Tool Box.

Editing Electrical Rules Class Data

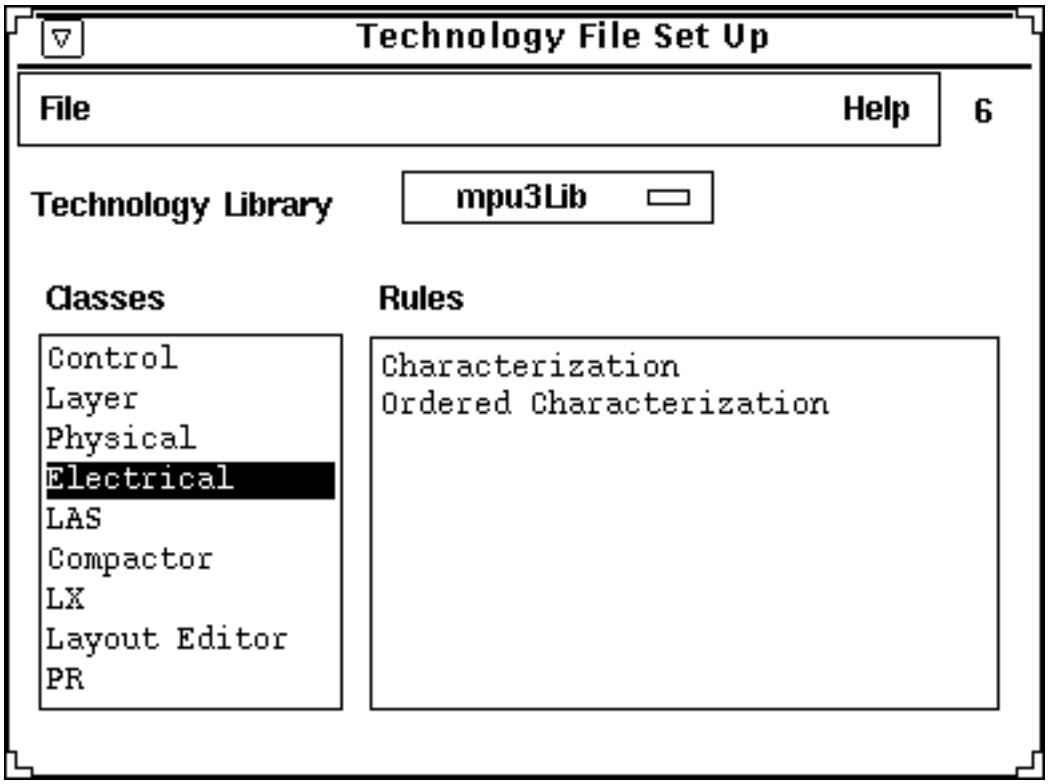
From the Technology File Tool Box, you can perform the following Electrical Rules class editing functions:

- Add a `characterizationRules` or `orderedCharacterizationRules` rule
- Delete a `characterizationRules` or `orderedCharacterizationRules` rule
- Edit the value type or value for an existing `characterizationRules` or `orderedCharacterizationRules` rule
- Edit or add a characterization rules table

To edit Electrical Rules class technology file data in virtual memory, perform the following steps:

1. From the Technology File Tool Box, choose *Edit Rules*.

   The Technology File Set Up form appears.



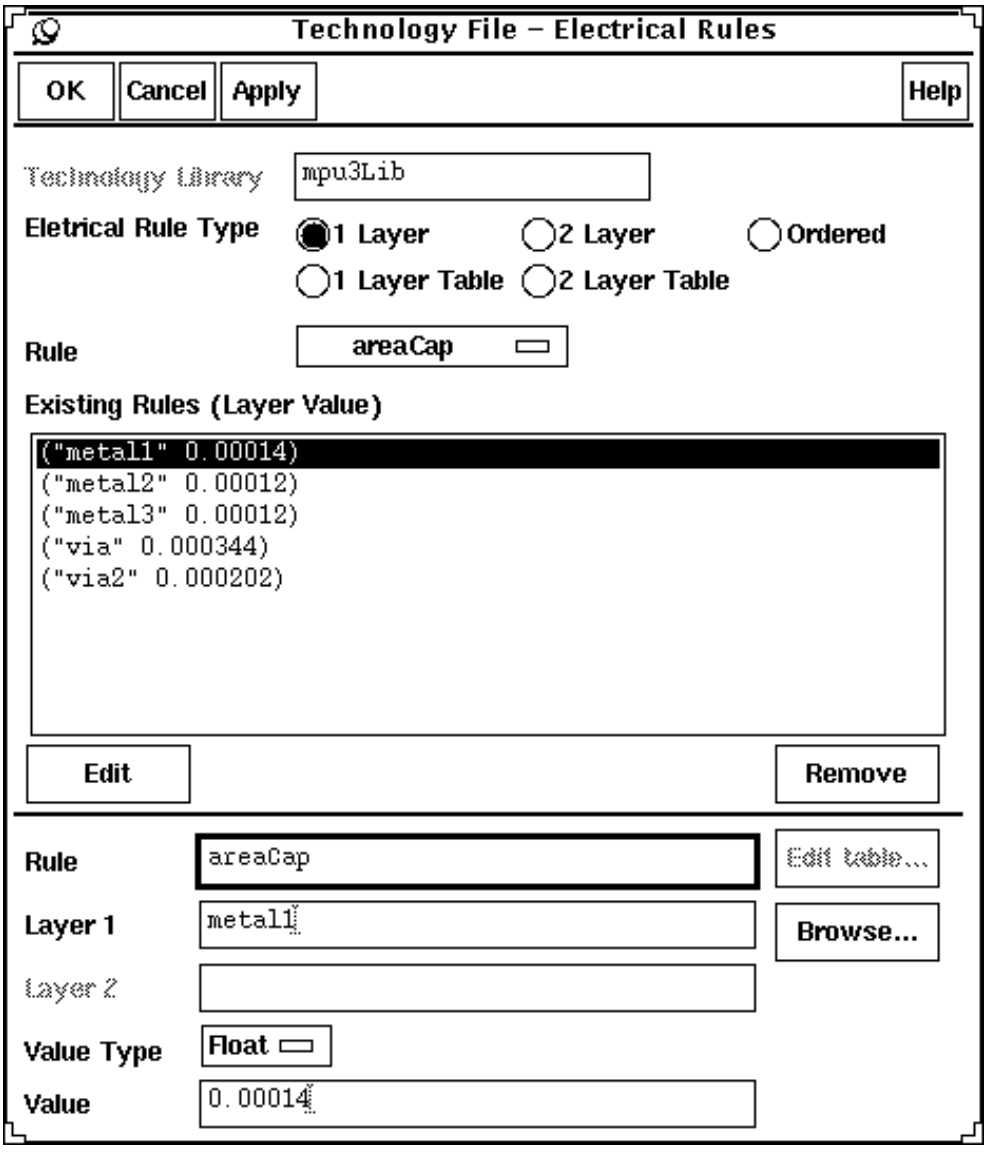
2. From the *Technology Library* cyclic field, choose the technology library containing the technology file to edit.

3. In the *Classes* list box, click on *Electrical* and, from the menu banner, choose *File – Edit*.

or

In the *Classes* list box, double-click on *Electrical*.

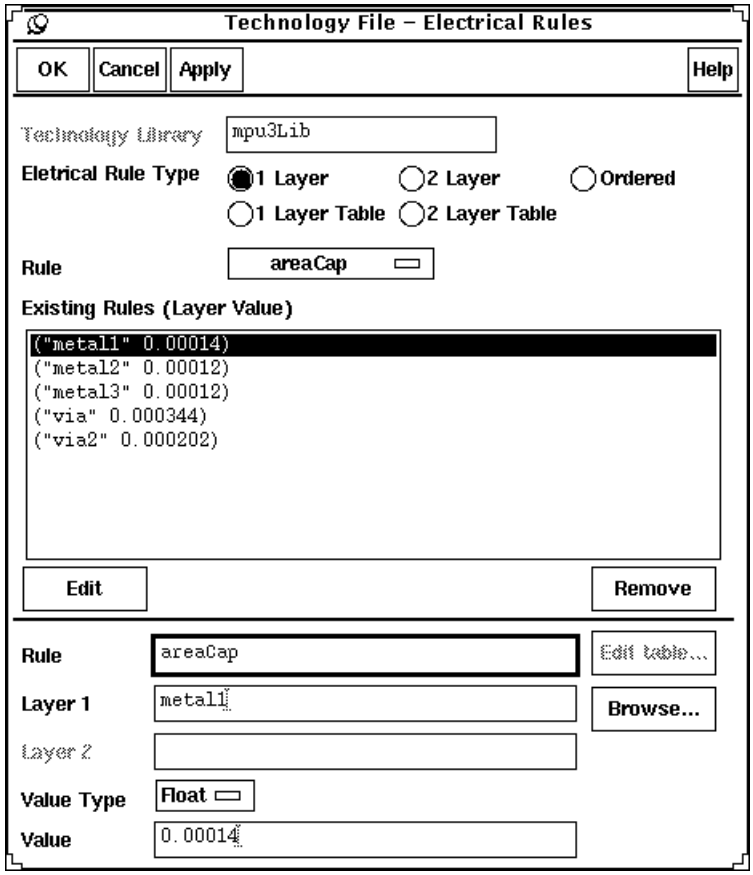
The Technology File – Electrical Rules form appears, displaying the Electrical Rules data currently in the technology file.



For a description of this form, see Appendix A.

4. Edit the data as described in the following:

To add a `characterizationRules` **or** `orderedCharacterization-Rules` **rule**, use either of the following methods:



Method 1

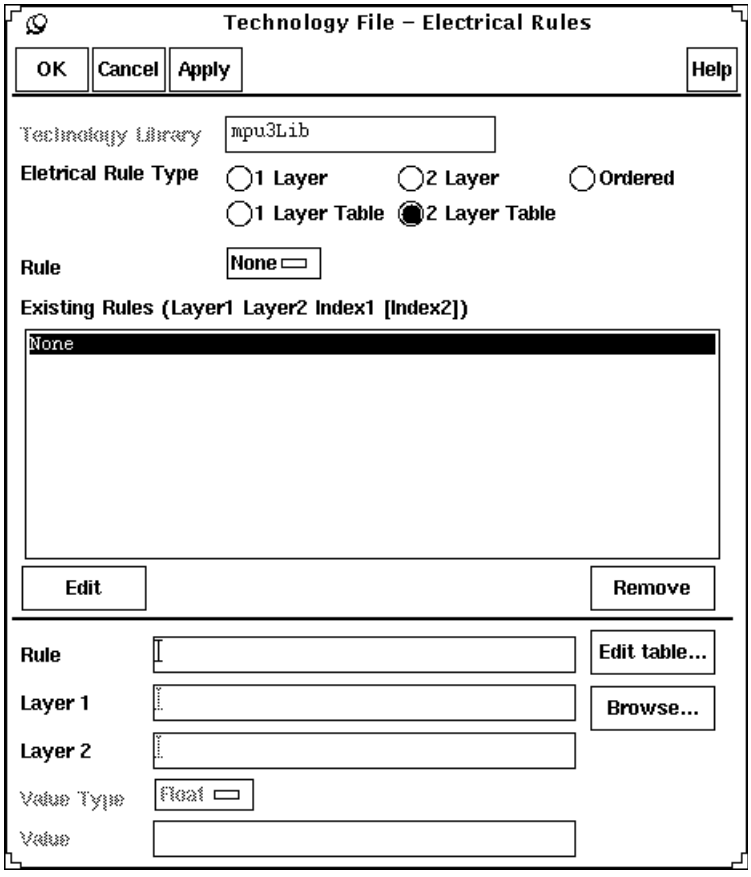
1. Click on the appropriate radio button to choose *1 Layer* to add a 1-layer characterization rule, *2 Layer* to add a 2-layer characterization rule, or *Ordered* to add an ordered characterization rule.
2. In the fields at the bottom of the form, type the arguments for the new rule and select the value 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.)
3. Click *Edit*. The software adds the new characterization rule to the *Existing Rules* list box.

Method 2

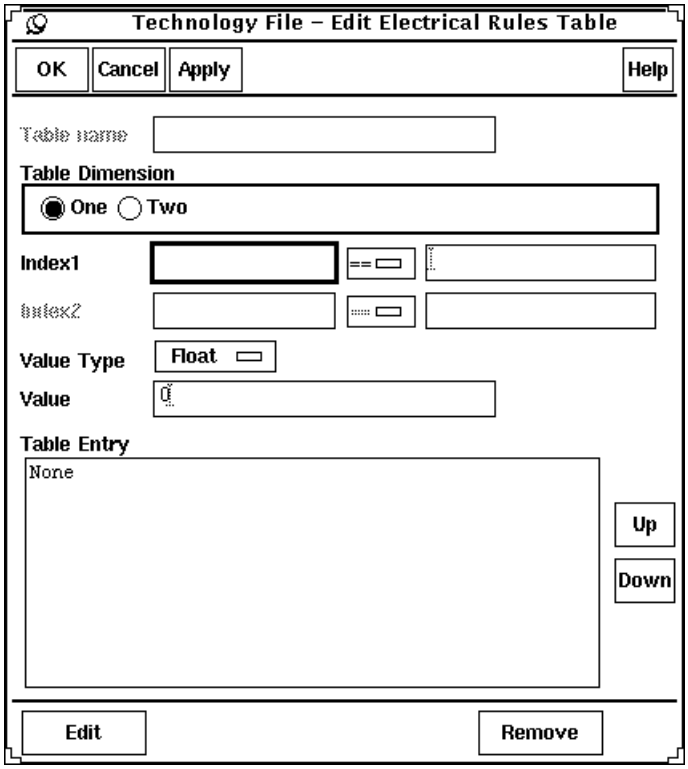
1. Click on the appropriate radio button to choose *1 Layer* or *2 Layer* to display `characterizationRules` or to choose *Ordered* to display `orderedCharacterizationRules` in the *Existing Rules* list box.
2. From the *Rule* cyclic field, choose the rule name. The software displays the rules of the 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. (You can use the Layer Browser as described in *Method 1*.)
5. Click *Edit*. The software adds the new characterization rule to the *Existing Rules* list box.

To add a `tableCharacterizationRules` rule, do the following:

1. Click on the appropriate radio button to choose *1 Layer Table* to add a 1-layer characterization rules table or *2 Layer Table* to add a 2-layer characterization rules table.
2. At the bottom of the form, in the *Rule* field, type the name of the rule table to create.
3. In the *Layer1* field, type the name of the layer for a 1-layer table or the first layer for a 2-layer table. For a 2-layer table, in the *Layer2* field, type the name of the second layer.
4. Click *Edit table*. The software displays the Technology File – Edit Electrical Rules Table form.



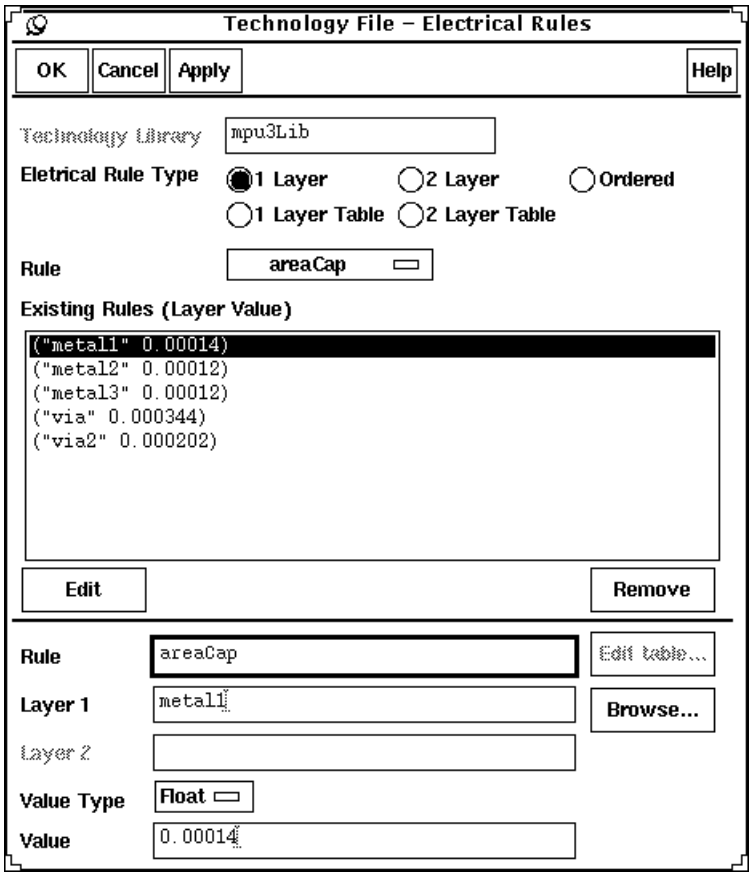
The Edit Electrical Rules Table form displays the rule name in the *Table name* field. For a description of this form, see Appendix A.



5. Click on the appropriate radio button to choose a one- or two-dimensional table.
6. In the first field of the *Index1* line, type the name of the first index. For a two-dimensional table, repeat for *Index2*.
7. From the match type cyclic field of the *Index1* line, choose the match type for the table entry index. For a two-dimensional table, repeat for *Index2*.
8. In the rightmost field of the *Index1* line, type the index to enter in the table entry. For a two-dimensional table, repeat for *Index2*.
9. From the *Value Type* cyclic field, choose the type for the value associated with the index or indices.
10. In the *Value* field, type the value to associate with the index or indices.
11. Click *Edit*. The software adds the table entry to the *Table Entry* list box.
12. Continue entering table entries until the table is complete.
13. When the table is complete, click *OK* to apply the table entries and dismiss the form.

To delete a table entry:
1. In the *Table Entry* list box, click on the table entry you want to delete.
2. Click *Remove*. The software removes the table entry from the *Table Entry* list box.

To delete an electrical rule:
1. Click on the appropriate radio button to choose *1 Layer* or *2 Layer* for `characterizationRules`, to choose *Ordered* for `orderedCharacterizationRules`, or to choose *1 Layer Table* or *2 Layer Table* for `tableCharacterizationRules`.
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 you want to delete.
4. Click *Remove*. 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`, to choose *Ordered* for `orderedCharacterizationRules`, or to choose *1 Layer Table* or *2 Layer Table* for `tableCharacterizationRules`.
2. From the *Rule* cyclic field, choose the rule name. For `characterizationRules` and `orderedCharacterizationRules`, the software displays the rules of the selected type with the selected rule name in the *Existing Rules* list box. For `tableCharacterizationRules`, the software displays the layer and index names for the table.
3. In the *Existing Rules* list box, click on the rule to edit. For `characterizationRules` and `orderedCharacterizationRules`,the software displays the rule arguments in the data fields at the bottom of the form. For `tableCharacterizationRules`, the software displays the rule name and layer name or names.
4. For `characterizationRules` and `orderedCharacterizationRules`, edit the data in the *Value Type* or *Value* fields. For `tableCharacterizationRules`, click *Edit table*, edit the table in the Edit Electrical Rules Table form, and click *OK* when done.
5. On the Technology File – Electrical Rules form, 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 *Save* from the Technology File Tool Box.

11

Editing Class Data through the Technology File Tool Box: Application-Specific Rules

This chapter discusses the following:

Editing Layout Editor Rules Class Data

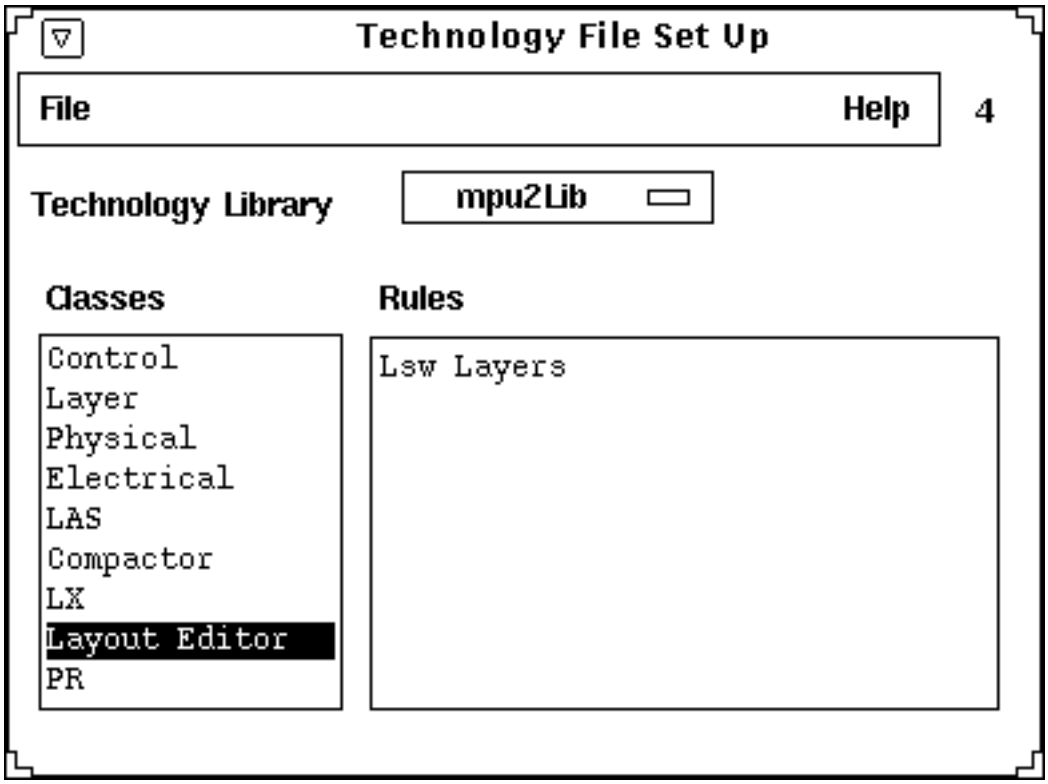
From the Technology File Tool Box, you can perform the following Layout Editor Rules class editing functions:

- Add a layer-purpose pair to the `leLswLayers` rule
- Delete a layer-purpose pair from the `leLswLayers` rule

To edit Layer Rules class technology file data in virtual memory, perform the following steps:

1. From the Technology File Tool Box, choose *Edit Rules*.

   The Technology File Set Up form appears.

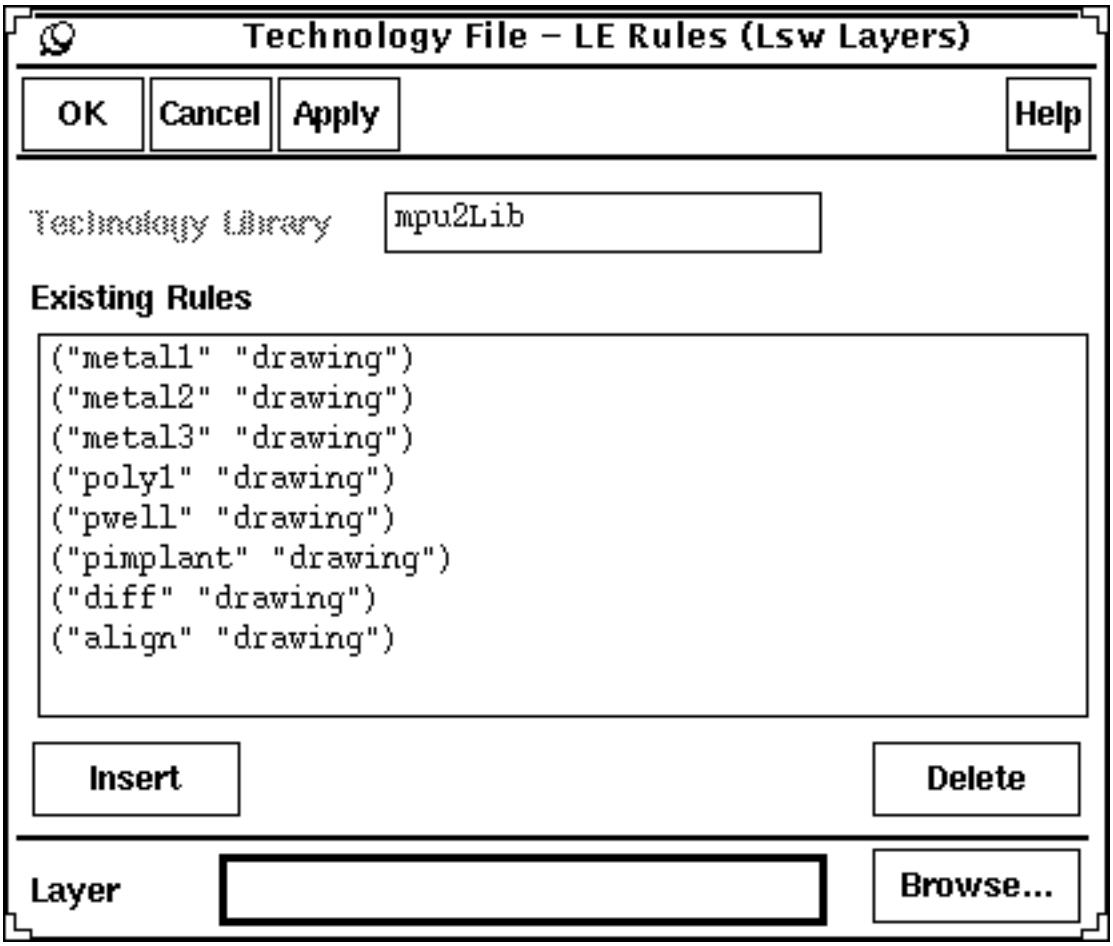


2. From the *Technology Library* cyclic field, choose the technology library containing the technology file to edit.
3. In the *Classes* list box, click on *Layout Editor* and, from the menu banner, choose *File – Edit*.

   or

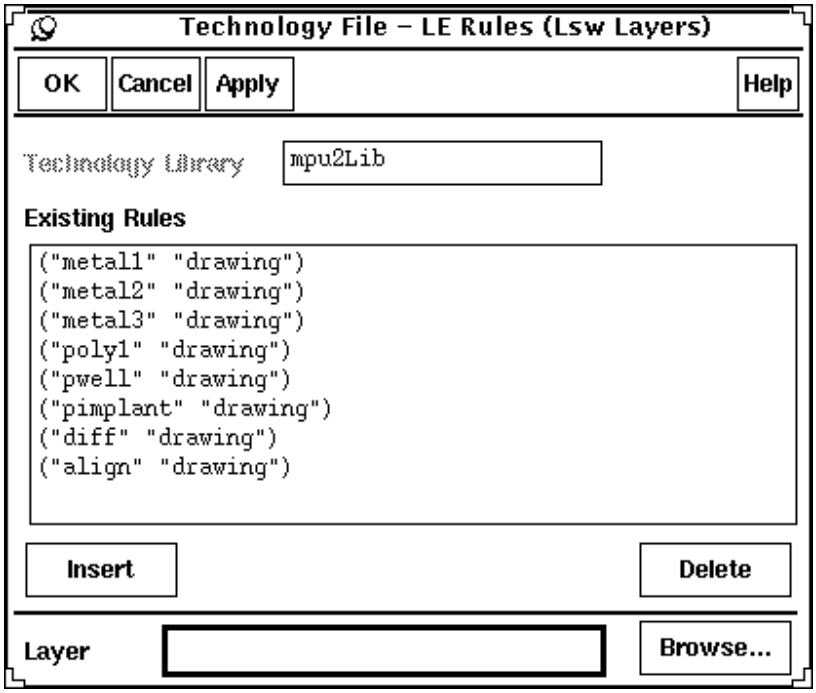
   In the *Classes* list box, double-click on *Layout Editor*.

The Technology File – LE Rules (Lsw Layers) form appears, displaying the Layout Editor Rules data for the `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 you want 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 Technology File Tool Box, choose *Save*.

Editing Virtuoso XL Rules Class Data

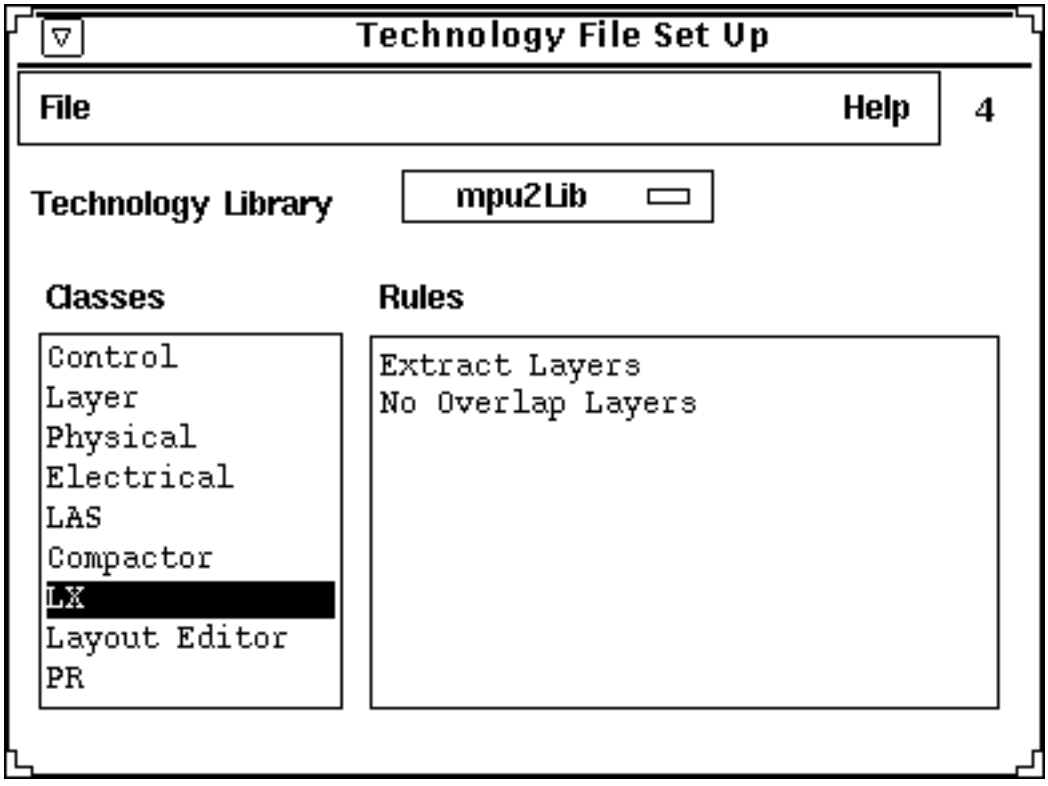
From the Technology File Tool Box, you can perform the following Virtuoso® XL Rules class editing functions:

- Add a layer to the `lxExtractLayers` rule
- Add an `lxNoOverlapLayers` rule
- Delete a layer from the `lxExtractLayers` rule
- Delete an `lxNoOverlapLayers` rule

To edit Layer Rules class technology file data in virtual memory, perform the following steps:

1. From the Technology File Tool Box, choose *Edit Rules*.

   The Technology File Set Up form appears.

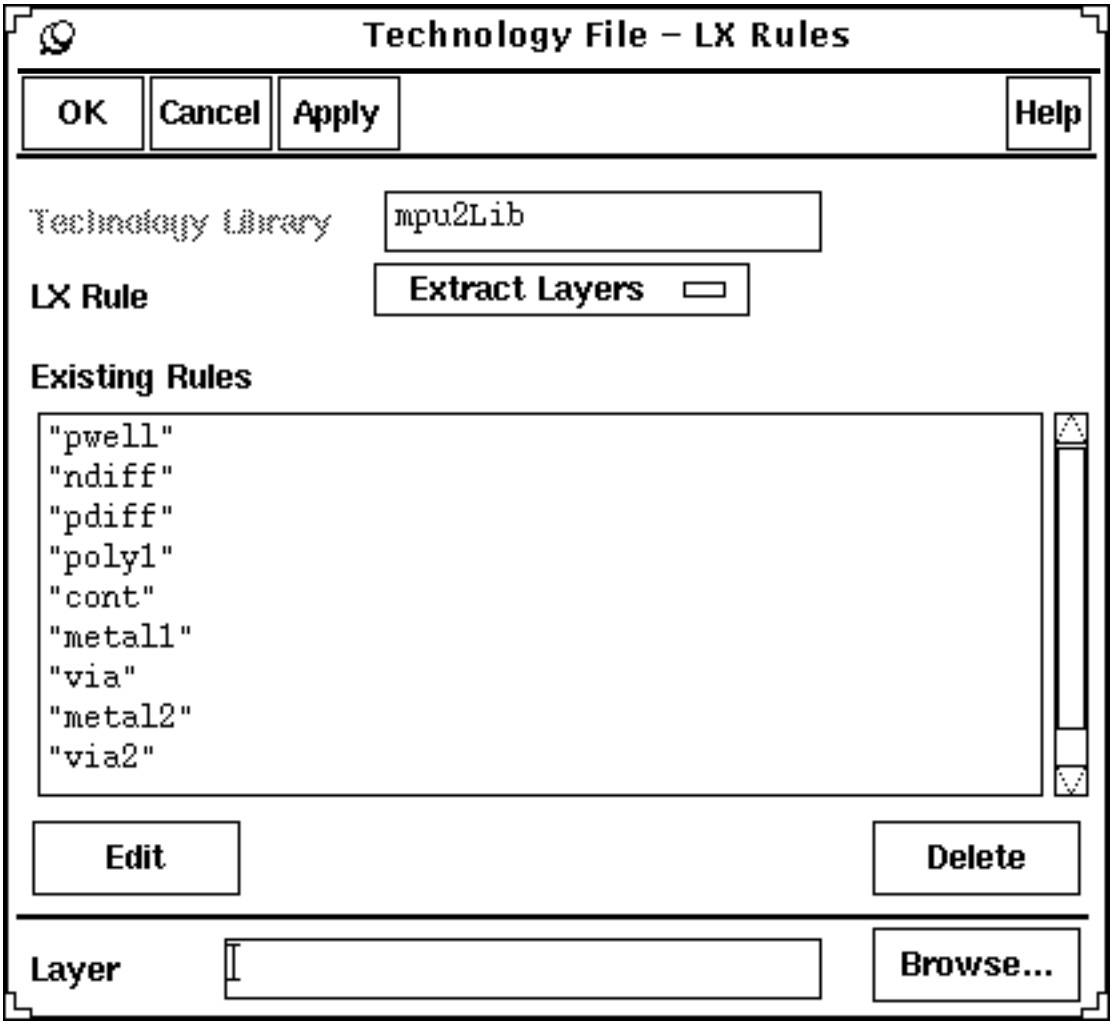


2. From the *Technology Library* cyclic field, choose the technology library containing the technology file to edit.
3. In the *Classes* list box, click on *LX* and, from the menu banner, choose *File – Edit*.

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.

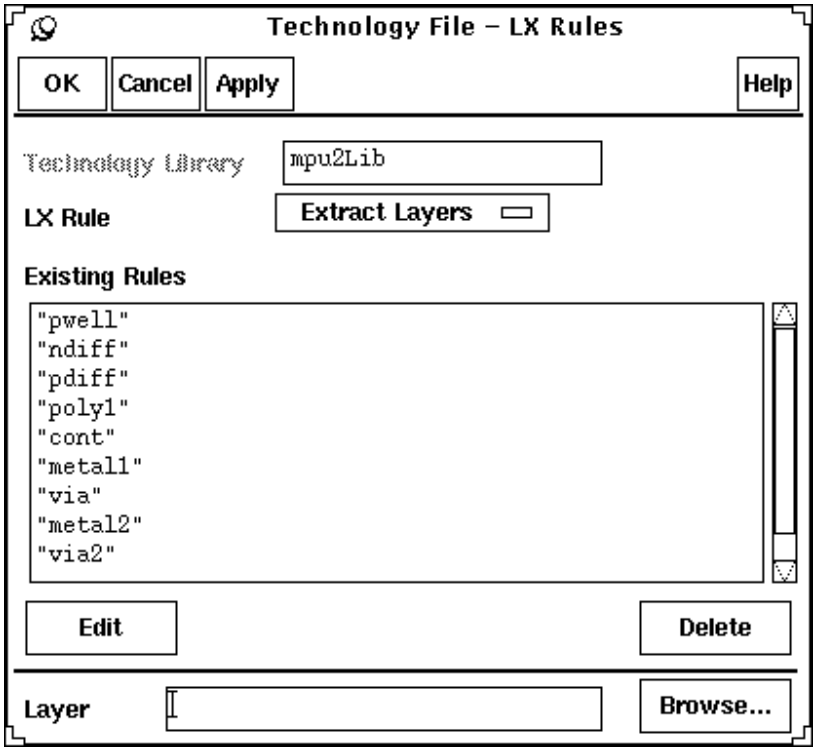
4. From the *LX Rule* cyclic field, choose the subclass to edit from the following:

- 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`



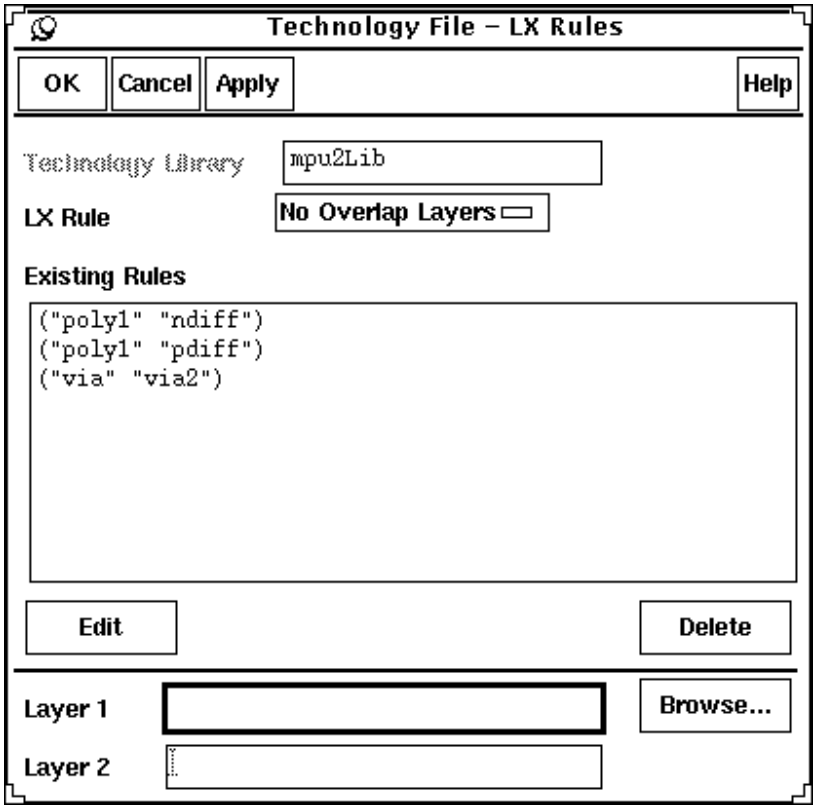
To add an `lxExtractLayers` layer, use either of the following methods:

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.

- `lxNoOverlapLayers`



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 you want 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 *Save* from the Technology File Tool Box.

Editing Virtuoso Compactor Rules Class Data

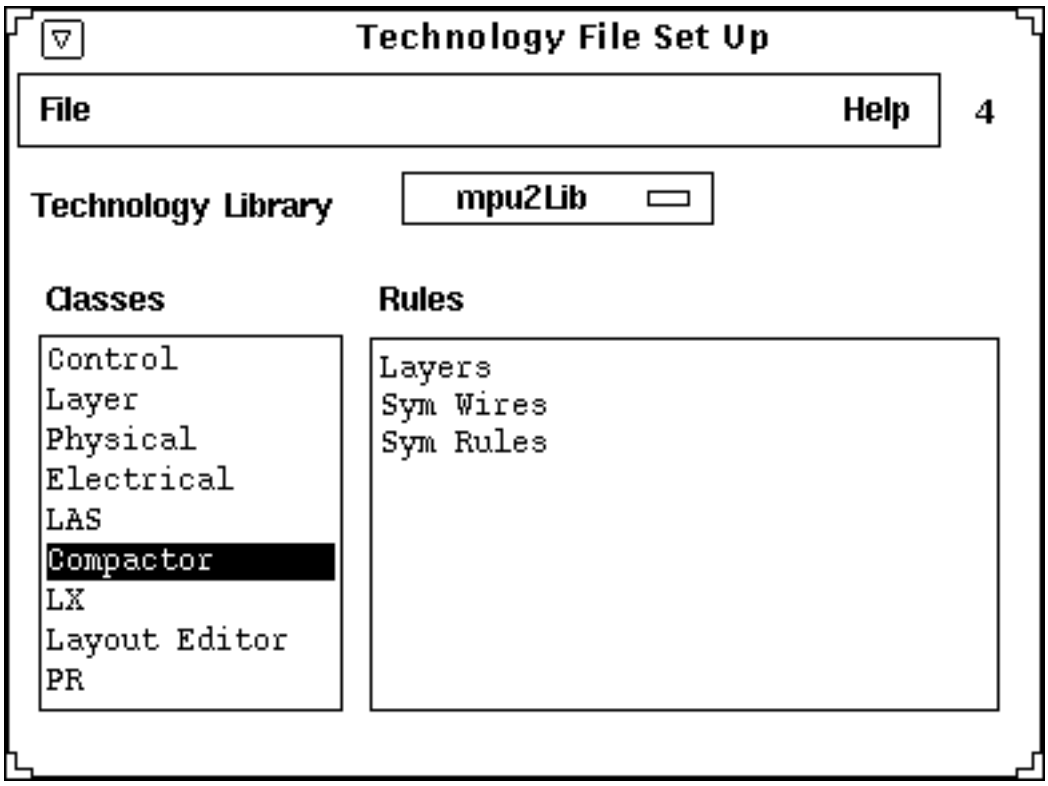
From the Technology File Tool Box, you can perform the following Virtuoso Compactor Rules class editing functions:

- 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 Technology File Tool Box, choose *Edit Rules*.

   The Technology File Set Up form appears.



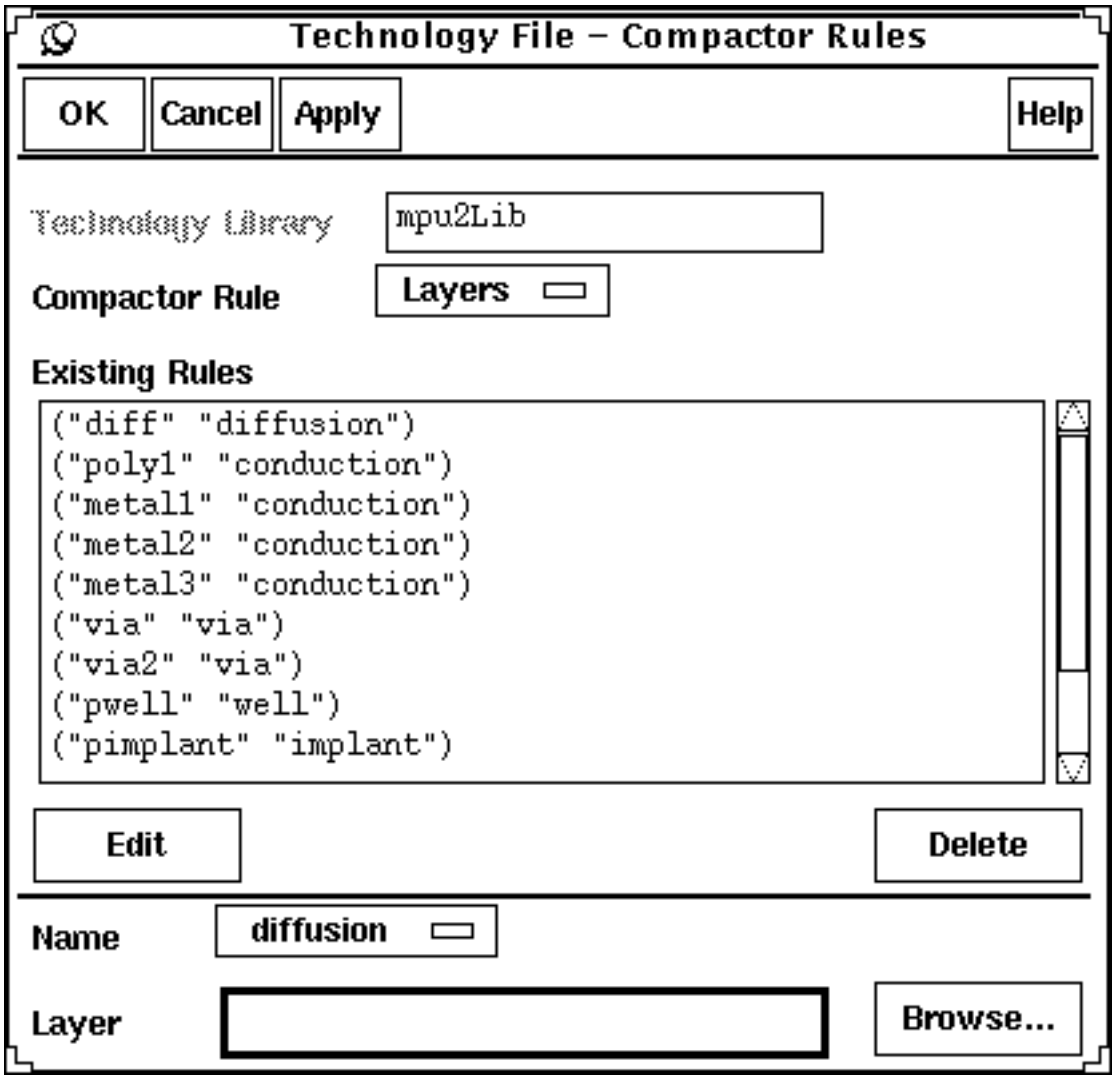
2. From the *Technology Library* cyclic field, choose the technology library containing the technology file to edit.

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.

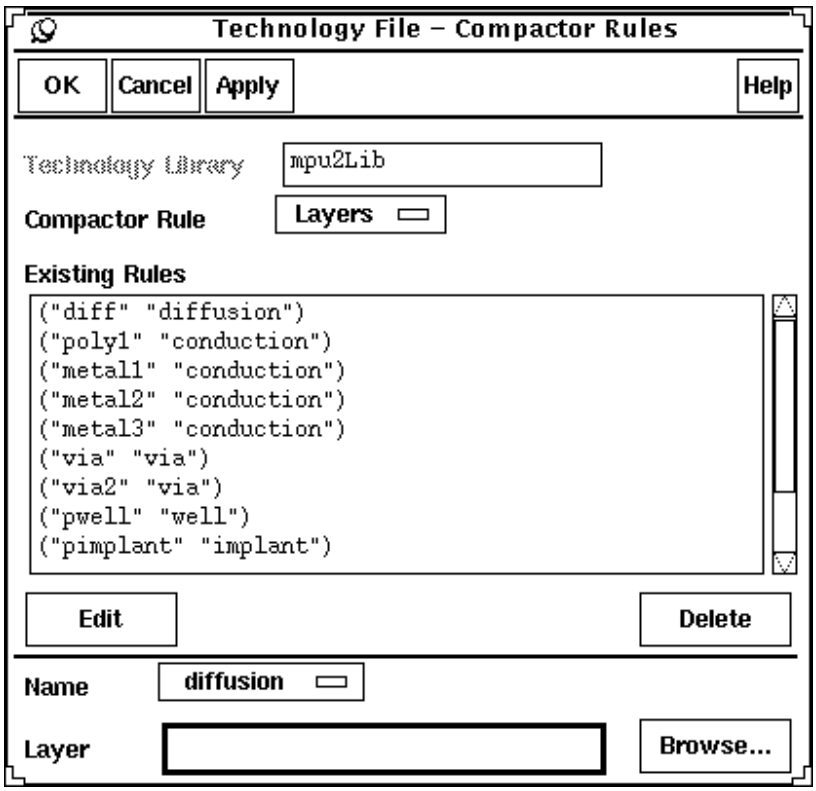
4. From the *Compactor Rule* cyclic field, choose the subclass to edit:

- ❑ 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 you want to delete.
2. Click *Delete*. The software removes the rule from the *Existing Rules* list box

❑ symWires

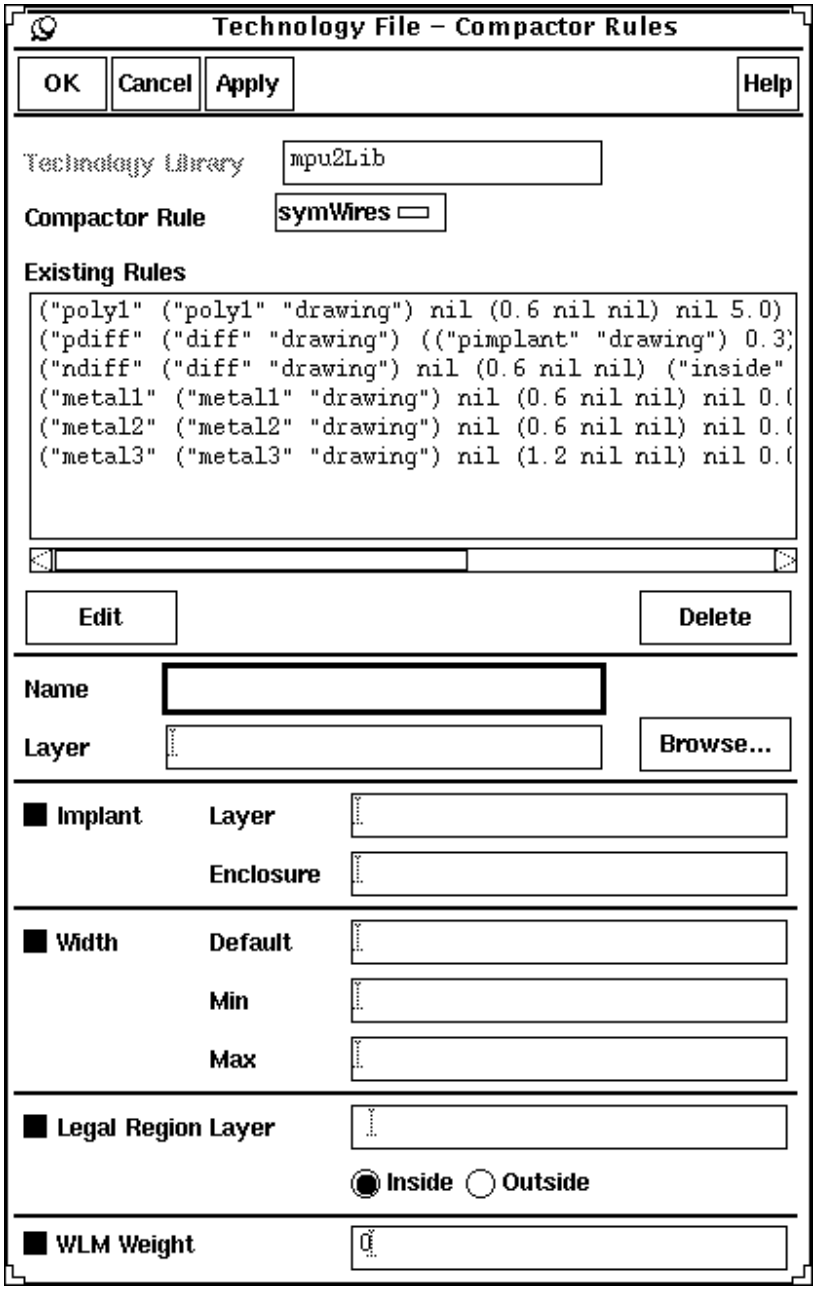
To add a `symWires` rule, use either of the following methods:

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*.)
Note: You must specify a new rule name to produce a new rule.
3. Click *Edit*. The software adds the new `symWires` rule to the *Existing Rules* list box.



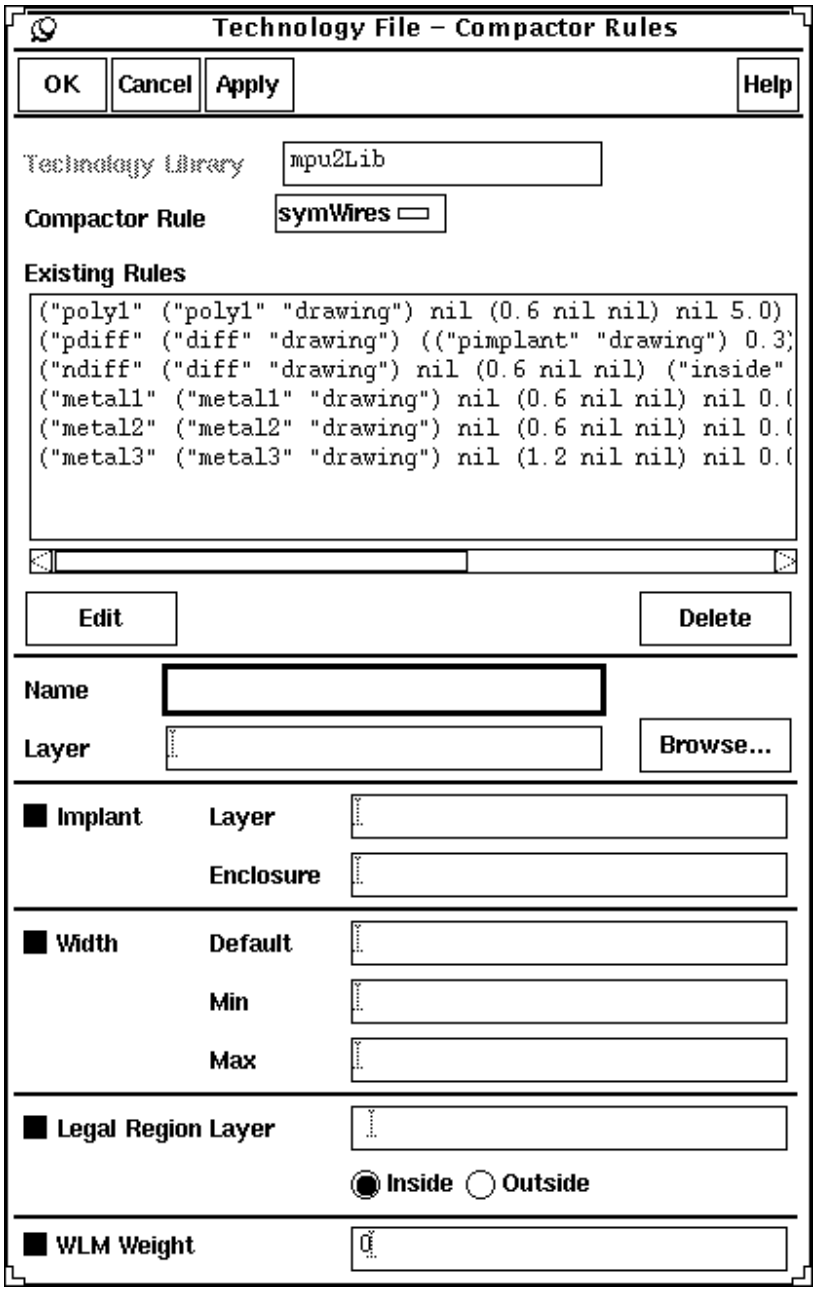
To delete a `symWires` rule:
1. In the *Existing Rules* list box, click on the rule you want to delete.
2. Click *Delete*. The software removes the rule from the *Existing Rules* list box.

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.

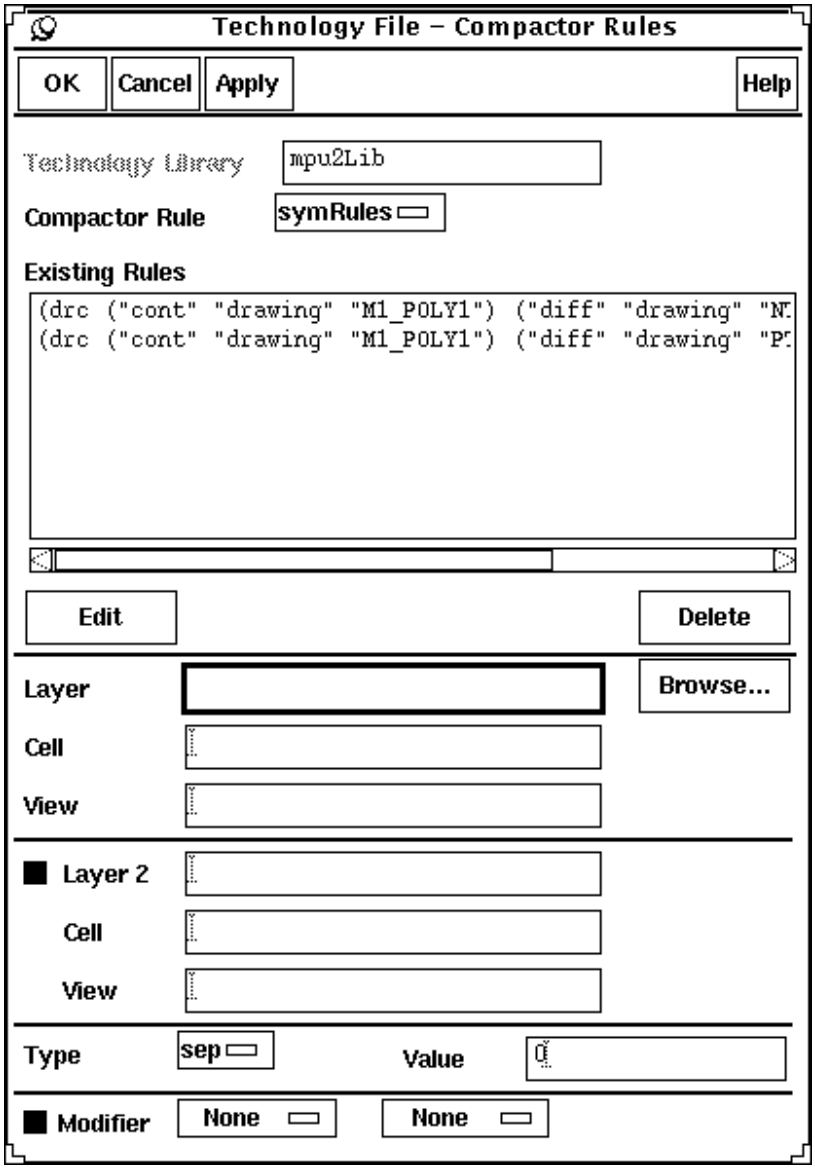


- `symRules`

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 you want 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 *Save* from the Technology File Tool Box.

Editing Place and Route Rules Class Data

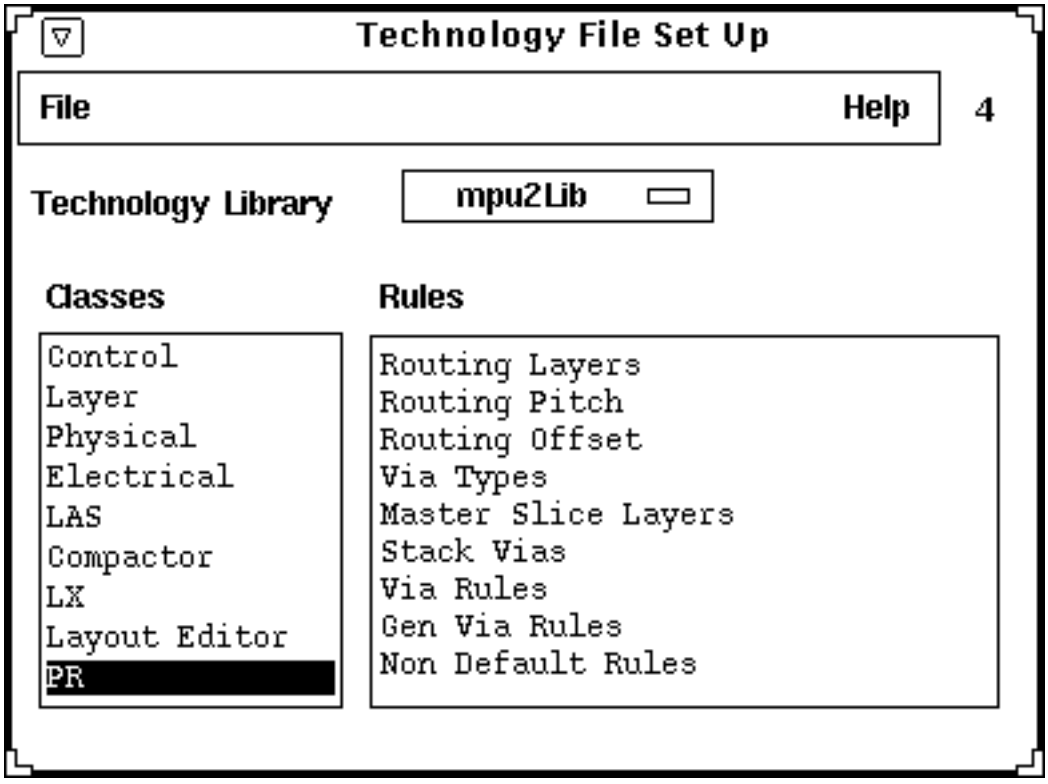
From the Technology File Tool Box, you can perform the following Place and Route Rules class editing functions:

- Add a `prRoutingLayers`, `prViaTypes`, `prStackVias`, `prMastersliceLayers`, `prViaRules`, `prGenViaRules`, or `prNonDefaultRules` rule
- Delete a `prRoutingLayers`, `prViaTypes`, `prStackVias`, `prMastersliceLayers`, `prViaRules`, `prGenViaRules`, or `prNonDefaultRules` rule
- Edit an existing `prRoutingLayers`, `prViaTypes`, `prStackVias`, `prMastersliceLayers`, `prViaRules`, `prGenViaRules`, or `prNonDefaultRules` rule

To edit Place and Route Rules class technology file data in virtual memory, perform the following steps:

1. From the Technology File Tool Box, choose *Edit Rules*.

   The Technology File Set Up form appears.

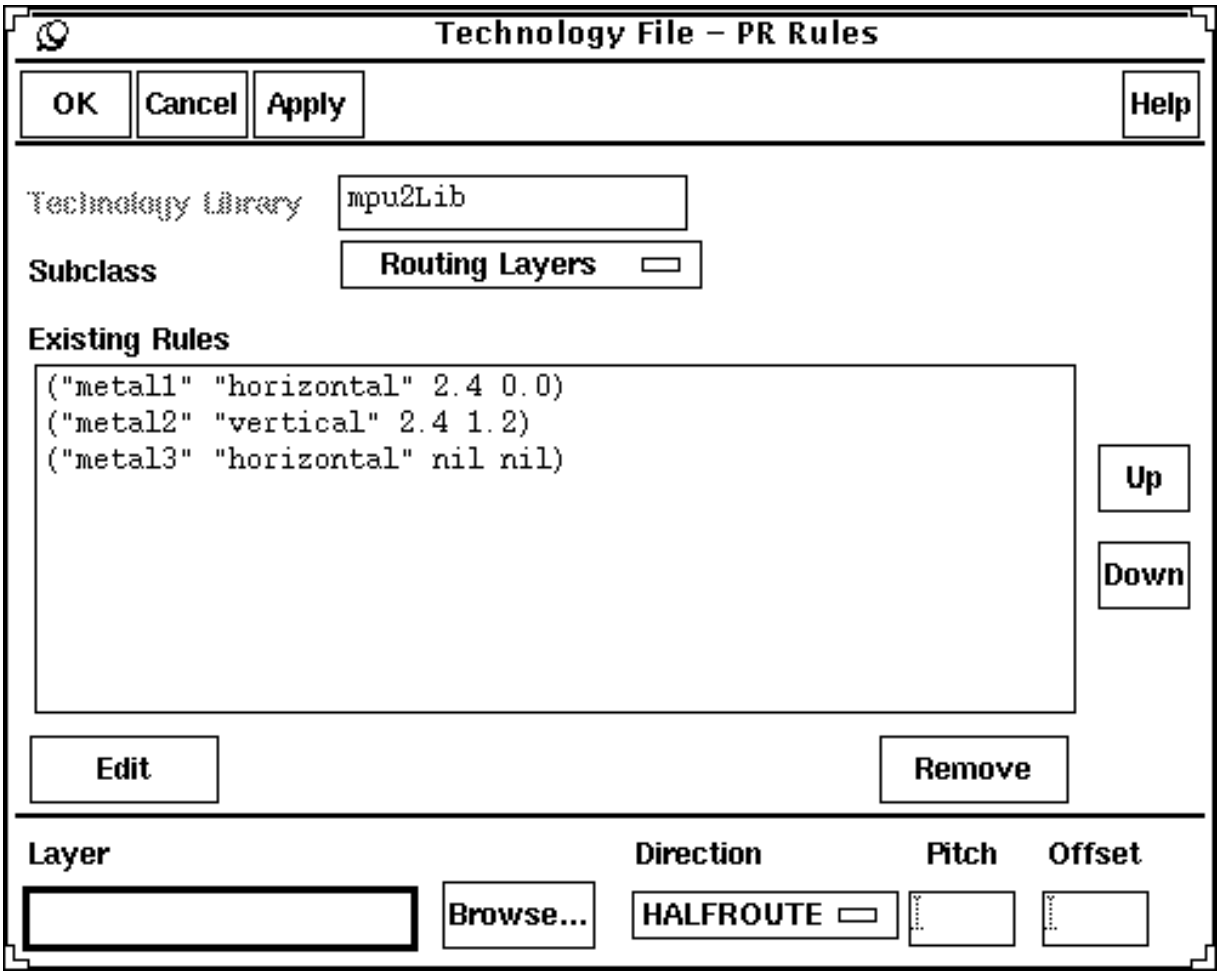


2. From the *Technology Library* cyclic field, choose the technology library containing the technology file to edit.
3. In the *Classes* list box, click on *PR* and, from the menu banner, choose *File – Edit*.

   or

   In the *Classes* list box, double-click on *PR*.

   The Technology File – PR Rules form appears, displaying the Place and Route Rules data currently in the technology file.



   For a description of this form, see Appendix A.

4. From the *Subclass* cyclic field, choose the subclass to edit:
   - ❑ Routing Layers, Routing Pitch, and Routing Offset
   - ❑ Via Types

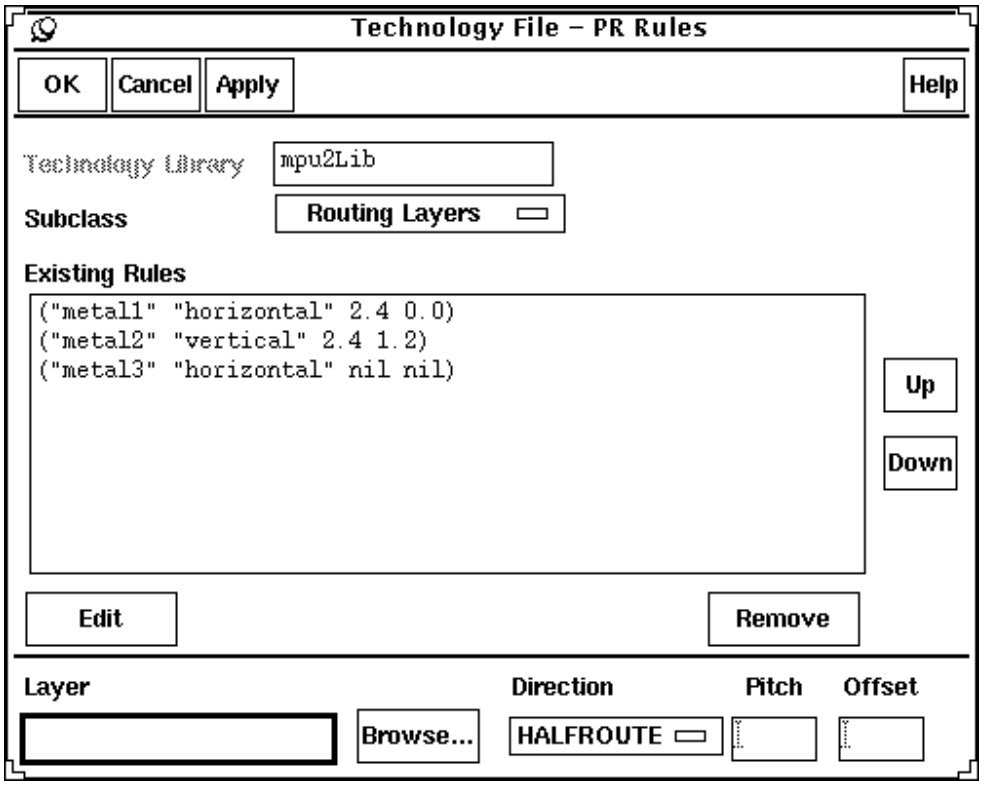
- Stack Vias
- Master Slice Layers
- Via Rules
- Generated Via Rules
- Nondefault Rules

The Technology File – PR Rules form displays the subclass data currently in the technology file.

5. Edit the data as described below:

- `prRoutingLayers`

To add a `prRouting-Layers`, `prRouting-Pitch`, and `prRouting-Offset` rule, use either of the following methods:



Method 1

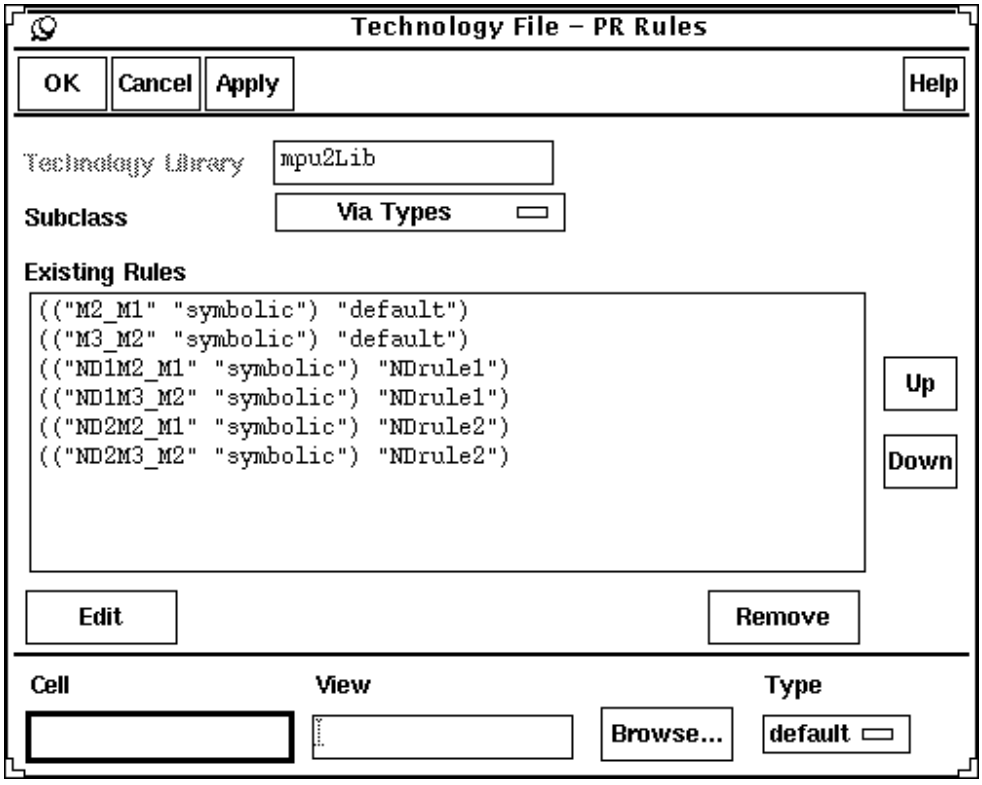
1. In the *Layer* field, type the layer name. (You can click *Browse* to view the names of the layers in the library with the Layer Browser. When you click on a layer in the Layer Browser, the software automatically loads it into the *Layer* field.)
2. From the *Direction* cyclic field, choose the routing direction keyword (*HALFROUTE*, *HORIZONTAL*, or *VERTICAL*).
3. In the *Pitch* field, type the pitch.
4. In the *Offset* field, type the offset.
5. Click *Edit*. The software adds the new `prRoutingLayers` rule, with data for the `prRoutingPitch` and `prRoutingOffset` as well, 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 `prRoutingLayers`, `prRoutingPitch`, and `prRoutingOffset` arguments in the *Layer*, *Direction*, *Pitch*, and *Offset* fields.
2. 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*.)
3. From the *Direction* cyclic field, choose the routing direction keyword (*HALFROUTE*, *HORIZONTAL*, or *VERTICAL*).
4. In the *Pitch* field, type the pitch.
5. In the *Offset* field, type the offset.
6. Click *Edit*. The software adds the new `prRoutingLayers` rule to the *Existing Rules* list box.

To delete a `prRoutingLayers`, `prRoutingPitch`, and `prRoutingOffset` rule:
1. In the *Existing Rules* list box, click on the entry defining the rules you want to delete.
2. Click *Remove*. The software removes the rules from the *Existing Rules* list box.

❑ prViaTypes



To add a `prViaTypes` rule, use either of the following methods:

Method 1

1. In the *Cell* field, type the cell name.
2. In the *View* field, type the view name. (You can click *Browse* to view the names of the cells and views in the library with the Library Browser. When you click on a layer in the Library Browser, the software automatically loads it into the *Cell* and *View* fields.)
3. From the *Type* cyclic field, choose the via type.
4. Click *Edit*. The software adds the new `prViaType` rule to the *Existing Rules* list box.

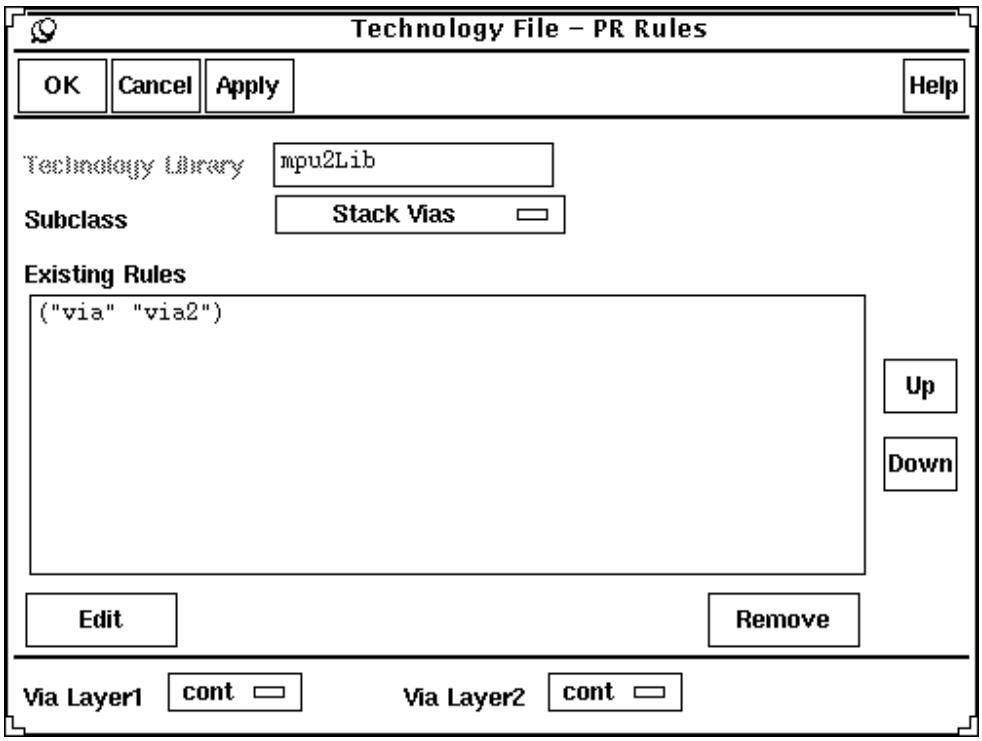
Method 2

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 `prViaType` arguments in the *Cell*, *View*, and *Type* fields.
2. In the *Cell* field, edit the cell name if you want a different cell name. (You can use the Library Browser as described in *Method 1*.)
3. In the *View* field, edit the view name if you want a different view name. (You can use the Library Browser as described in *Method 1*.)
4. From the *Type* cyclic field, choose the via type.
5. Click *Edit*. The software adds the new `prViaType` rule to the *Existing Rules* list box.

To delete a `prViaType` rule:
1. In the *Existing Rules* list box, click on the rule you want to delete.
2. Click *Remove*. The software removes the rule from the *Existing Rules* list box.

❑ prStackVias

To add a `prStackVias` rule, use either of the following methods:



Method 1

1. From the *Via Layer1* cyclic field, choose the first stack layer name.
2. From the *Via Layer2* cyclic field, choose the second stack layer name.
3. Click *Edit*. The software adds the new `prStackVias` rule to the *Existing Rules* list box.

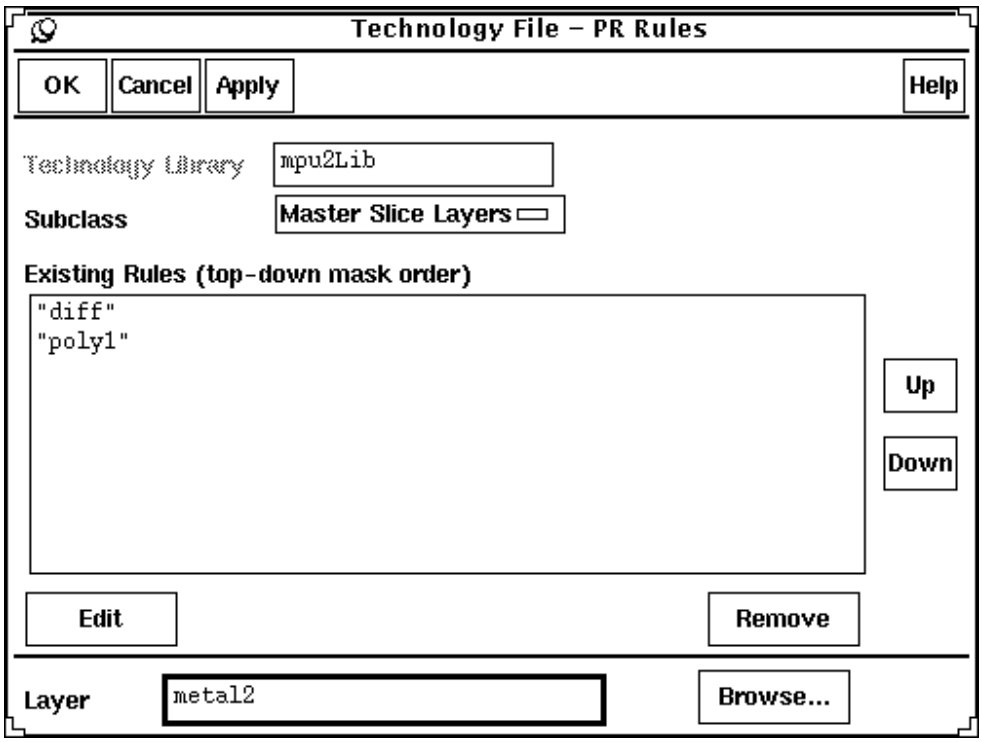
Method 2

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 `prStackVias` layers in the two cyclic fields.
2. From the *Via Layer1* cyclic field, choose the first stack layer name if it is different from the one already selected.
3. From the *Via Layer2* cyclic field, choose the second stack layer name if it is different from the one already selected.
4. Click *Edit*. The software adds the new `prStackVias` rule to the *Existing Rules* list box.

To delete a `prStackVias` rule:
1. In the *Existing Rules* list box, click on the rule you want to delete.
2. Click *Remove*. The software removes the rule from the *Existing Rules* list box.

- `prMastersliceLayers`

To add a `prMasterslice-Layers` rule, use either of the following methods:



Method 1

1. 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.)

2. Click *Edit*. The software adds the new `prMastersliceLayers` 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 `prMastersliceLayers` layer in the *Layer* field.

2. In the *Layer* field, edit the layer name. (You can use the Library Browser as described in *Method 1*.)

3. Click *Edit*. The software adds the new `prMastersliceLayers` rule to the *Existing Rules* list box.

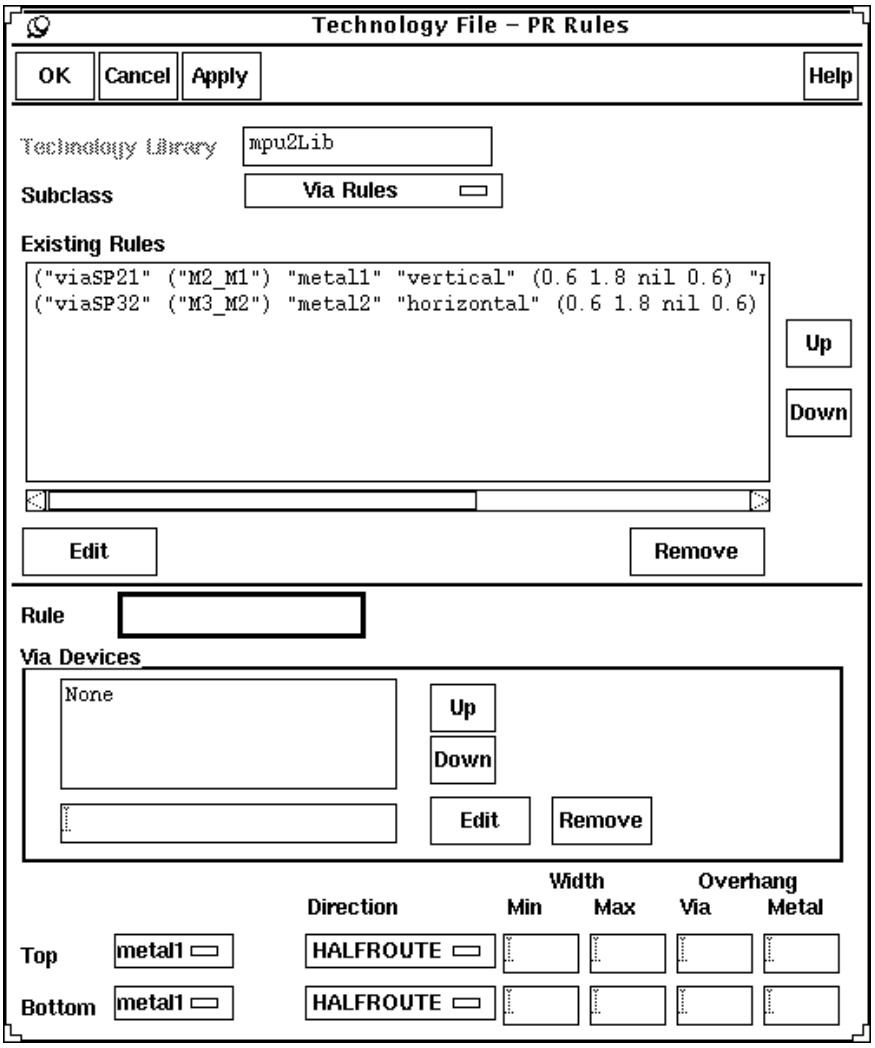
To delete a `prMastersliceLayers` rule:

1. In the *Existing Rules* list box, click on the rule you want to delete.

2. Click *Remove*. The software removes the rule from the *Existing Rules* list box.

❑ prViaRules

To add a `prViaRules` rule, use either of the following methods:



Method 1

1. In the *Rule* field, type the rule name.
2. For each via to use with the rule: In the *Via Devices* field beneath the *Via Devices* list box, type the via name and click *Edit* to the right of the field. The software adds the via device to the list box.
3. From the *Top* cyclic field, choose the first layer.
4. From the *Bottom* cyclic field, choose the second layer.
5. From the *Direction* cyclic fields, choose the direction for each layer.
6. In the *Width* and *Overhang* fields, type the minimum and maximum width and overhang for each layer.
7. Click *Edit*. The software adds the new `prViaRules` rule to the *Existing Rules* list box.

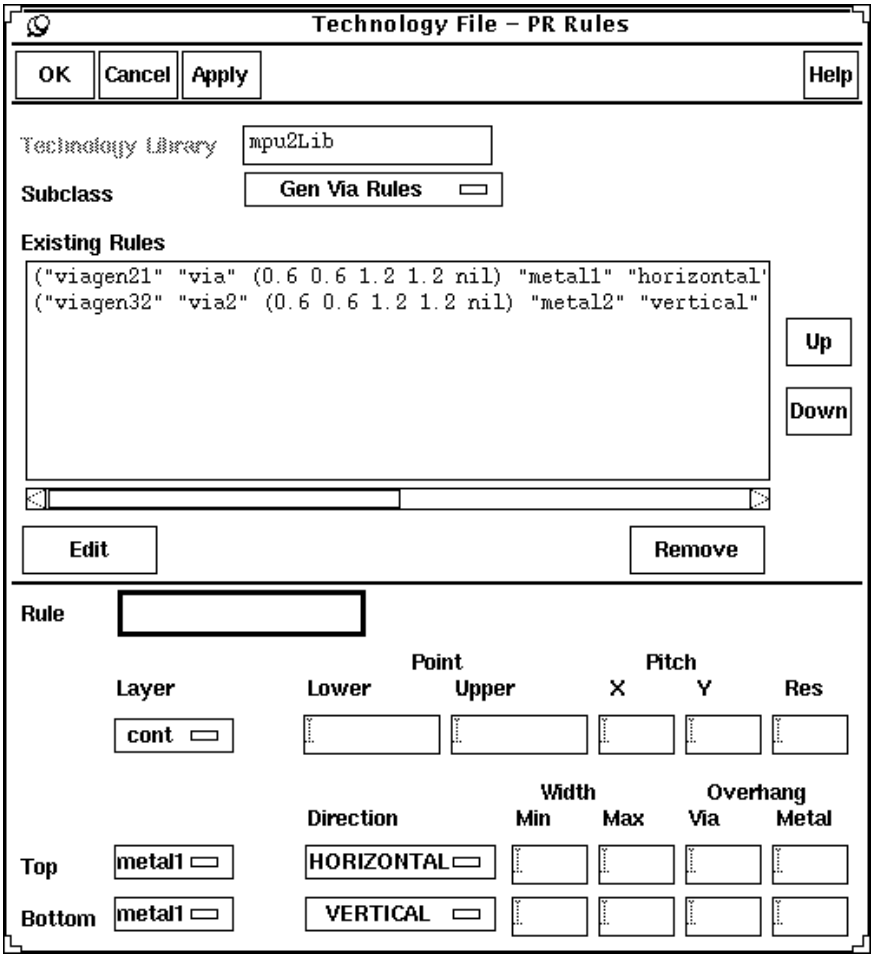
Method 2

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 `prViaRules` data in the fields in the rule-defining section of the form.
2. Edit the rule data for the new rule.
3. Click *Edit*. The software adds the new `prViaRules` rule to the *Existing Rules* list box.

To delete a `via device when defining a rule`:
1. In the *Via Devices* list box, click on the device you want to delete.
2. Click *Remove*. The software removes the device from the *Via Devices* list box.

To delete a `prViaRules` rule:
1. In the *Existing Rules* list box, click on the rule you want to delete.
2. Click *Remove*. The software removes the rule from the *Existing Rules* list box.

❑ prGenViaRules



To add a `prGenViaRules` rule, use either of the following methods:

Method 1

1. In the *Rule* field, type the rule name.
2. From the *Layer* cyclic field, choose the layer.
3. In the *Point Lower* field, type the lower point of the bounding box for the via layer.
4. In the *Point Upper* field, type the upper point of the bounding box for the via layer.
5. In the *Pitch* fields, type the x pitch and the y pitch.
6. From the *Top* cyclic field, choose the first layer.
7. From the *Bottom* cyclic field, choose the second layer.
8. From the *Direction* cyclic fields, choose the direction for each layer.
9. In the *Width* and field, type the minimum and maximum widths allowed for each wire, in user units.
10. In the *Overhang Via* field, type the minimum spacing between the contact cut and the outer edge of the via.
11. In the *Overhang Metal* field, type the minimum overhang of the via to the wire.
12. Click *Edit*. The software adds the new `prViaRules` rule to the *Existing Rules* list box.

Method 2

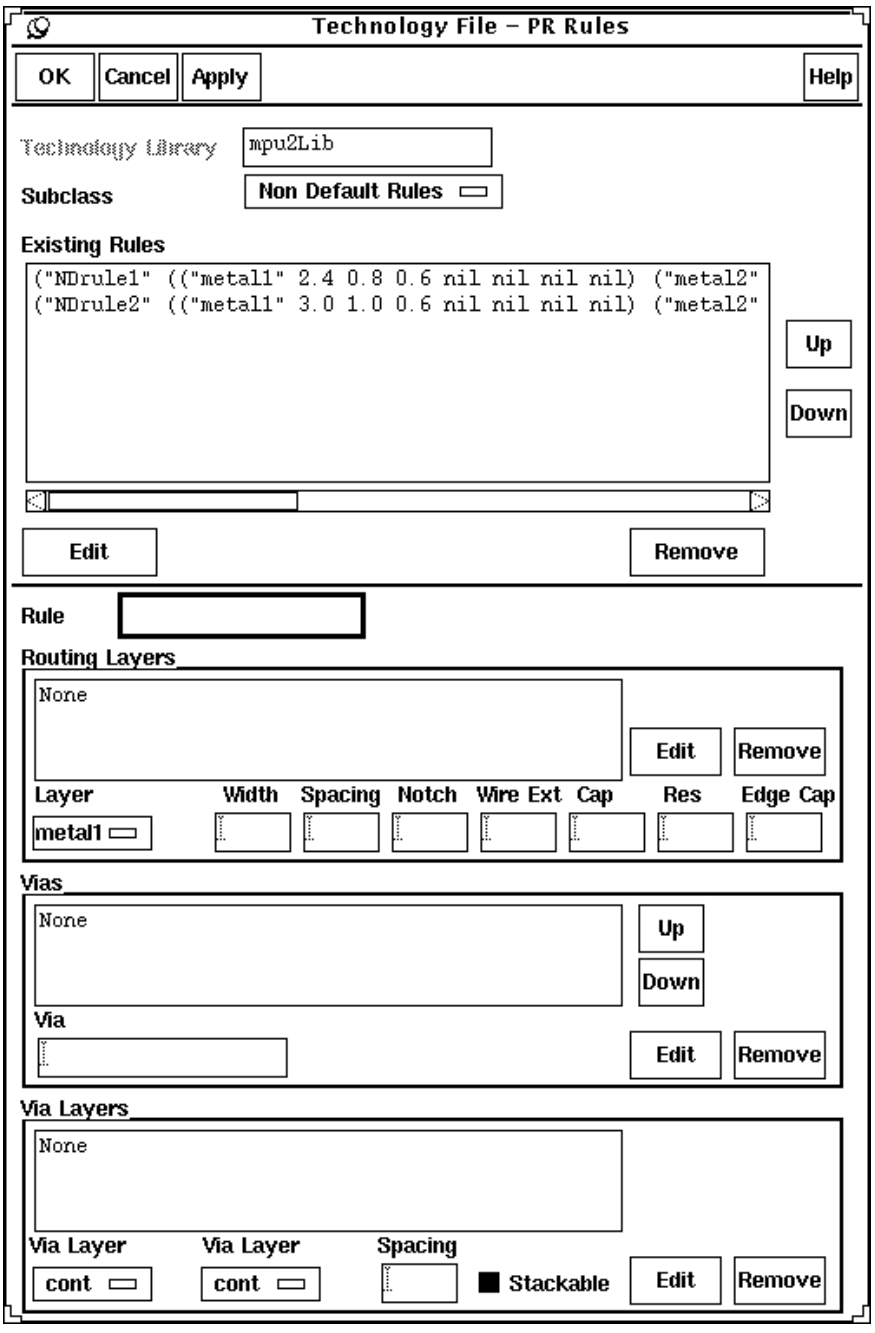
1. In the *Existing Rules* list box, click on a rule to use as a basis to edit to define a new rule. The form displays the `prGenViaRules` data in the fields in the rule-defining section of the form.
2. Edit the rule data for the new rule.
3. Click *Edit*. The software adds the new `prViaRules` rule to the *Existing Rules* list box.

To delete a `prGenViaRules` rule:

1. In the *Existing Rules* list box, click on the rule you want to delete.
2. Click *Remove*. The software removes the rule from the *Existing Rules* list box.

❑ prNonDefaultRules

To add a `prNonDefaultRules` rule, use either of the following methods:



Method 1

1. In the *Rule* field, type the rule name.

2. For each routing layer to use with the rule, in the *Routing Layers* section of the form:

a. From the *Layer* cyclic field, choose the layer.

b. In the *Width* field, type the width of the layer.

c. In the *Spacing* field, type the minimum spacing allowed for the layer.

d. In the *Notch* field, type the notch spacing allowed for the layer.

e. In the *WireExt* field, type the wire extension allowed for the layer.

f. In the *Cap* field, type the capacitance allowed for the layer.

g. In the *Res* field, type the resistance allowed for the layer.

h. In the *Edge Cap* field, type the edge capacitance allowed for the layer.

i. Click *Edit* next to the *Routing Layers* list box. The software adds the new layer to the list box.

3. For each via to use with the rule, in the *Vias* section of the form:

a. In the *Via* field, type the via name.

b. Click *Edit* in the *Vias* section. The software adds the new via to the *Vias* list box.

4. For each via layer to use with the rule, in the *Via Layers* section of the form:

a. From the *Via Layer* cyclic fields, choose the via layers.

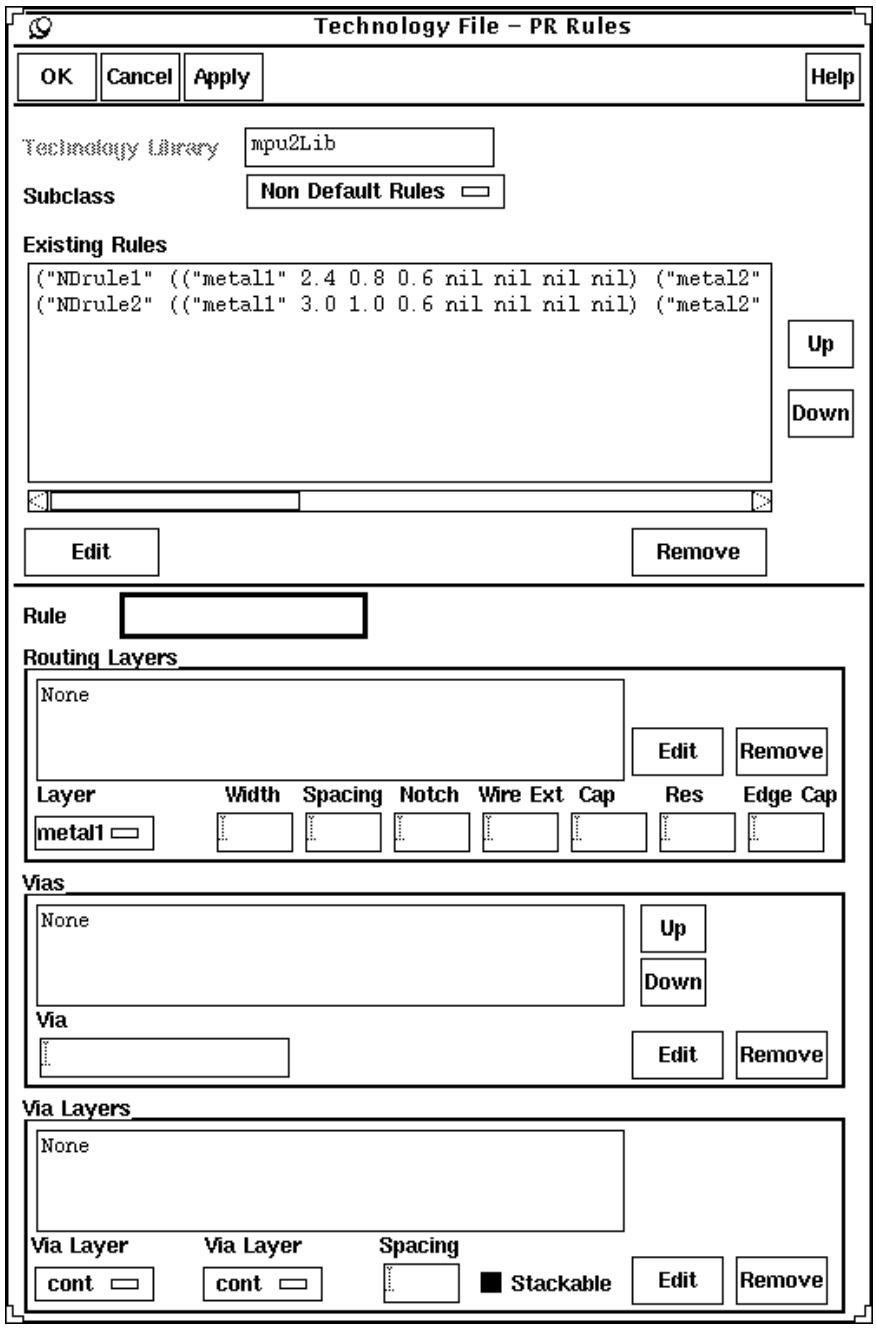
b. In the *Vias* field, type the minimum spacing allowed between the via layers.

c. Click *Edit* in the *Via Layers* section. The software adds the new via layers to the *Vias* list box.

5. Click *Edit* in the top section of the form. The software adds the new `prNonDefaultRules` 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 `prNonDefaultRules` data in the fields in the rule-defining sections of the form.
2. Edit the rule data for the new rule.
3. Click *Edit* in the top section of the form. The software adds the new `prViaRules` rule to the *Existing Rules* list box.



To delete a `routing layer when defining a rule`:
1. In the *Routing Layers* list box, click on the layer you want to delete.
2. Click *Remove*. The software removes the layer from the *Routing Layers* list box.

To delete a `via when defining a rule`:
1. In the *Vias* list box, click on the via you want to delete.
2. Click *Remove*. The software removes the via from the *Vias* list box.

To delete a `via layer when defining a rule`:
1. In the *Via Layers* list box, click on the via you want to delete.
2. Click *Remove*. The software removes the via layer from the *Vias* list box.

To delete a `prNonDefaultRules` rule:
1. In the *Existing Rules* list box, click on the rule you want to delete.
2. Click *Remove*. The software removes the rule from the *Existing Rules* list box.

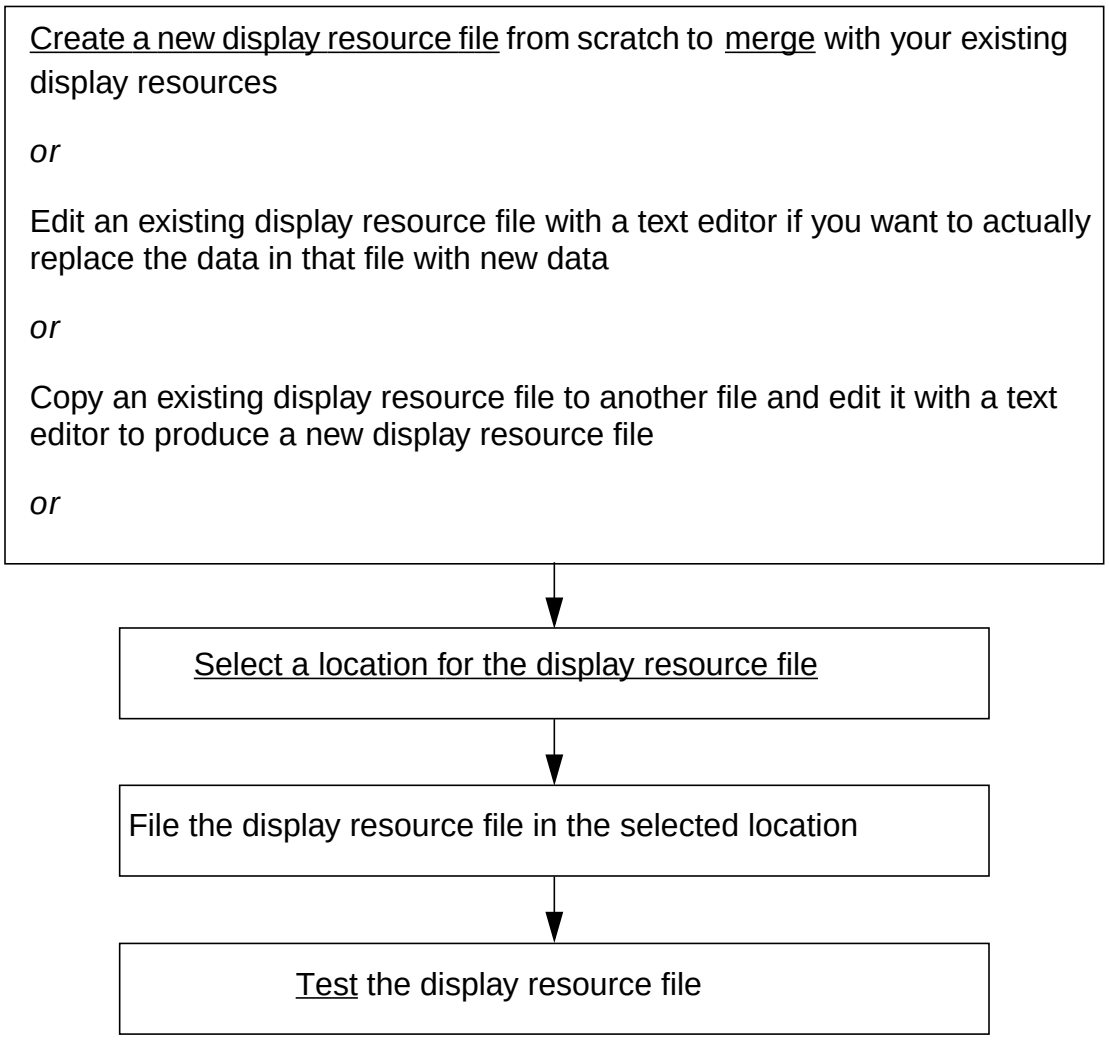
12

Editing, Reusing, and Merging Display Resources

This chapter discusses the following:

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:



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 Display Resources Tool Box” on page 302.

- Design Framework II (DFII) only: Use Cadence® SKILL language functions to load a display resource file and edit display resources in virtual memory. See “Editing Display Resource Data with SKILL Functions (Design Framework II Only)” on page 328 for a summary of the 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. Be aware, however, of how it will be used when you choose a place to file it. See “How Cadence Design Software Handles Multiple Display Resource Files” on page 33 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. →

```
firstlib/display.drf
    ( display   white       255  255  255 )
    ( display   silver      217  230  255 )
    ( display   cream       255  255  204 )
```

Second file merged. When this file is merged, its definition of `silver` overwrites the definition of `silver` from the first file. →

```
secondlib/display.drf
    ( display   silver      220  233  252 )
    ( display   pink        255  191  242 )
    ( display   magenta     255  0    254 )
```

Third file merged. When this file is merged, its definition of `magenta` overwrites the definition of `magenta` from the second file. →

```
thirdlib/display.drf
    ( display   magenta     255  0    255 )
```

Resultant merged file:

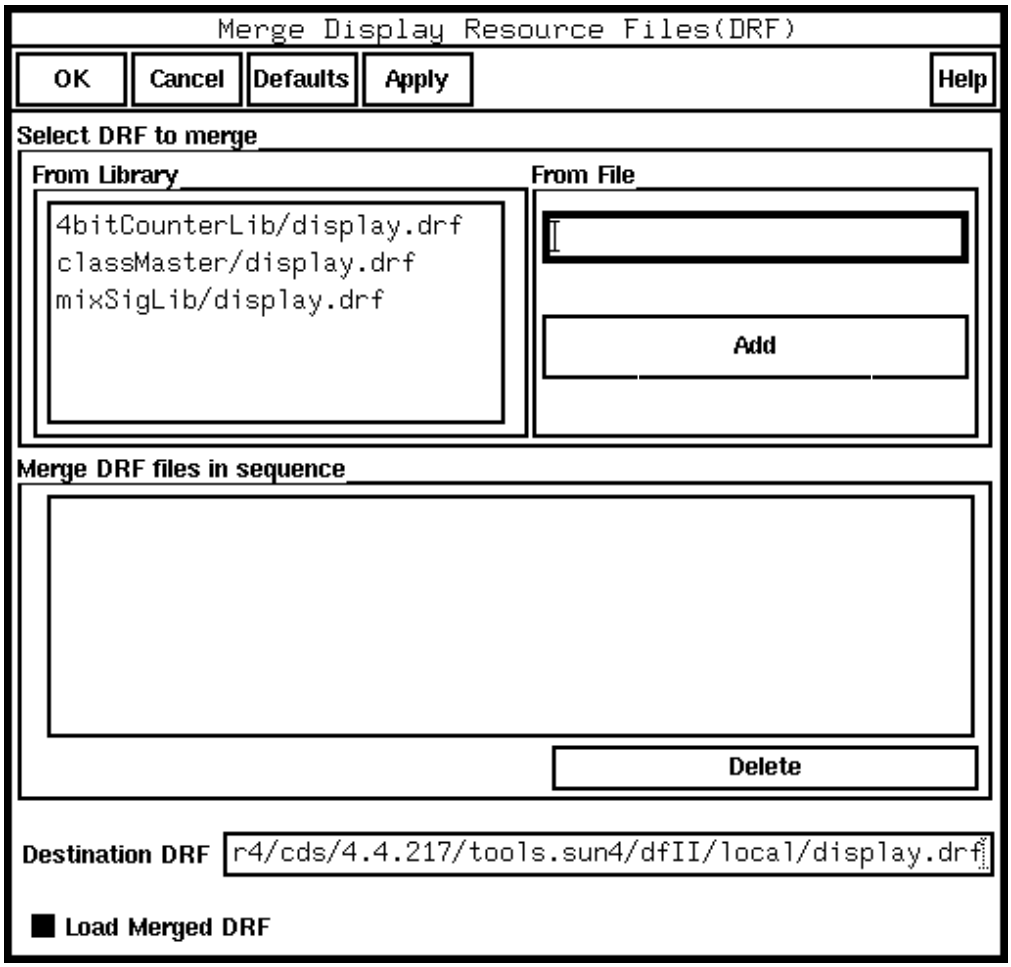
```
mergedlib/display.drf
    ( display   white       255  255  255 )
    ( display   cream       255  255  204 )
    ( display   silver      220  233  252 )
    ( display   pink        255  191  242 )
    ( display   magenta     255  0    255 )
```

Merging Multiple Display Resource Files into One File

To merge multiple display resource files, perform the following steps:

1. From the Display Resources Tool Box, choose *Merge*.

   The Merge Display Resource Files (DRF) form appears.



   For a description of this form, see Appendix A.

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 Cadence Design Software Handles Multiple Display Resource Files" on page 33 for more information.

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

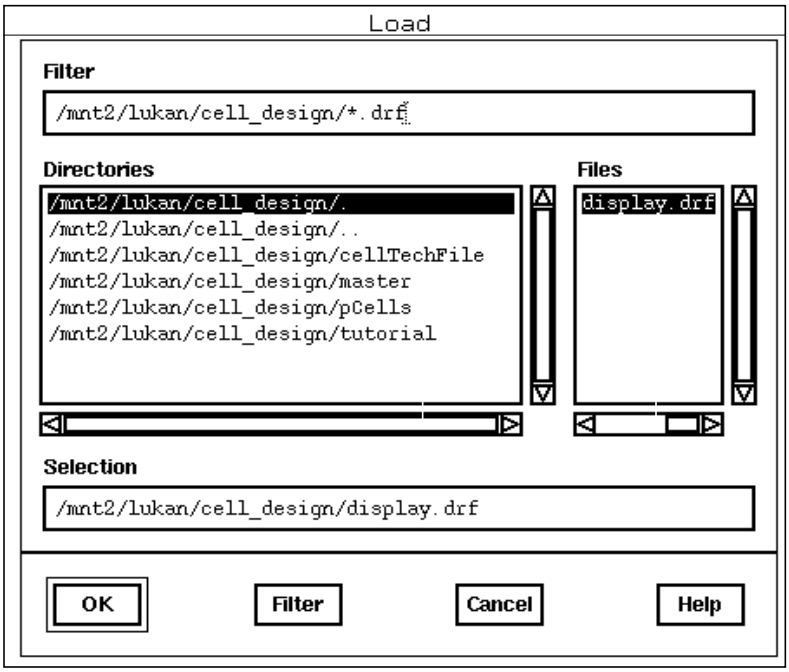
When you load a display resource file, the data in the file is merged with the data already in virtual memory. To merge new display resource data into the display resource data in virtual memory, perform the following steps:

1. From the Display Resources Tool Box, choose *Edit*.

   The Display Resource Editor form appears.

2. From the Display Resource Editor menu banner, choose *File – Load*.

   The Load form appears.



For a description of this form, see Appendix A.

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 330 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*.

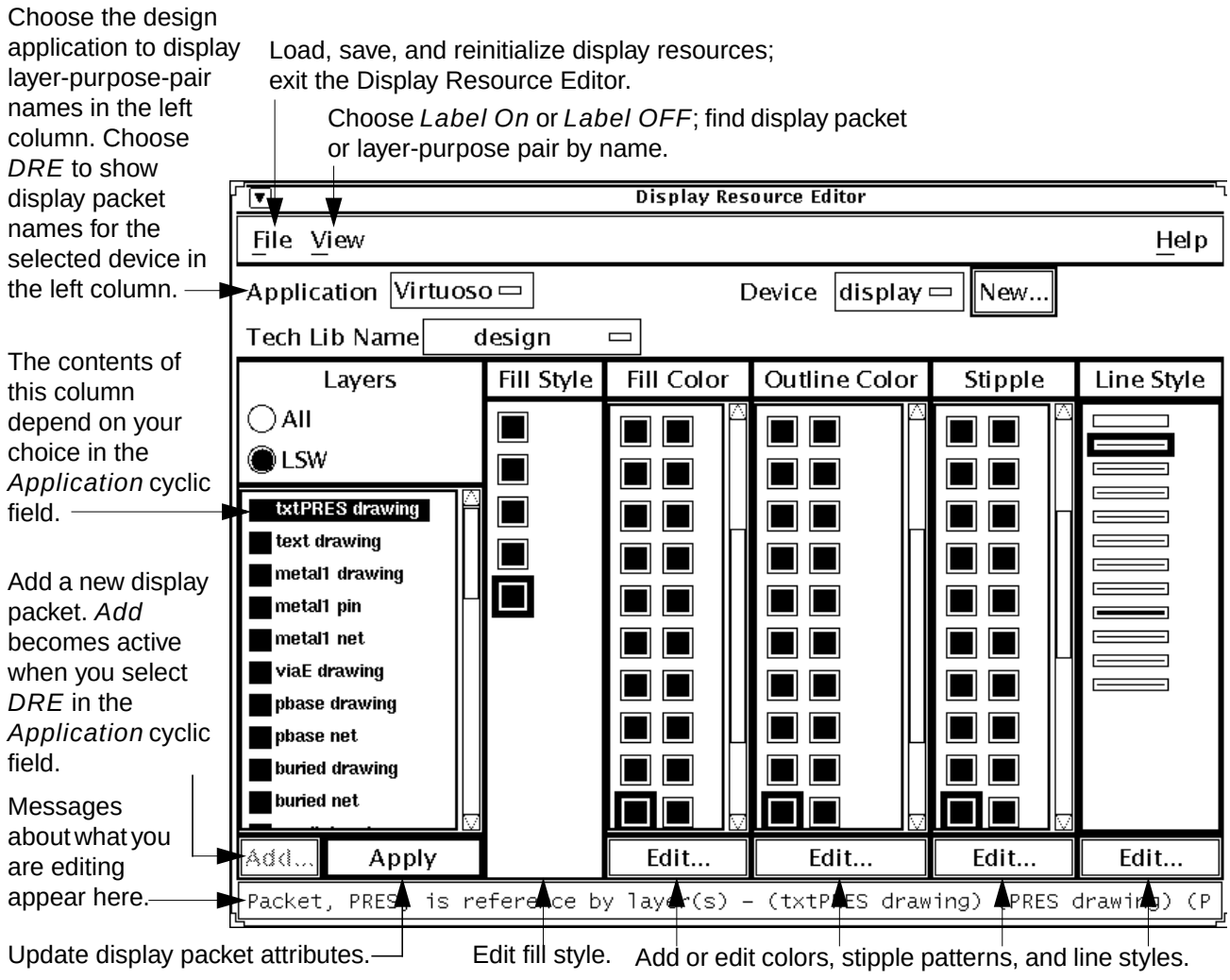
Editing Display Resource Data through the Display Resources Tool Box

The Display Resource Editor, which is accessible through the Display Resources Tool Box, 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 Cadence Design Software Handles Multiple Display Resource Files" on page 33 for details.

To access the Display Resource Editor,

➤ Choose *Edit* from the Display Resources Tool Box.

The Display Resource Editor appears, with the current application selected in the *Application* cyclic field. For a description of this form, see Appendix A.



Display Resource Data Accessible through the Display Resources Tool Box

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 search and editing functions:

- Find a display packet by name
- Find a layer-purpose pair by name
- Change a display packet definition
- Change layer display
- Add colors, stipple patterns, and line styles
- Edit colors, stipple patterns, and line styles
- Add a display packet
- Add a display device

You cannot do the following with the Display Resource Editor:

- Delete a display packet
- Delete a device, color, stipple, or line style

To perform these functions, you must edit the `display.drf` file with a text editor or, for the delete functions only and in DFII only, edit the display resource data in virtual memory with SKILL functions.

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

Finding a Display Packet or a Layer-Purpose Pair by Name

The Display Resource Editor allows you to find a display packet or a layer-purpose pair by name.

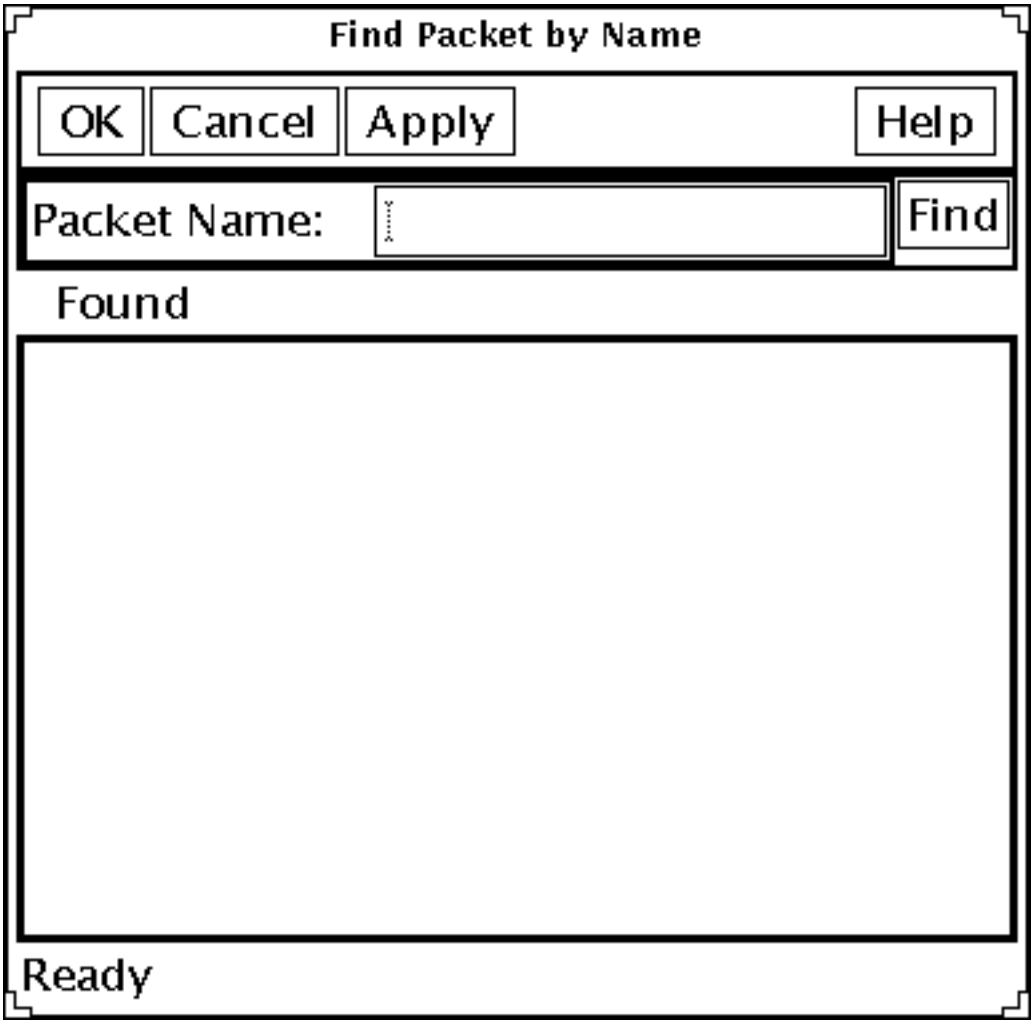
Finding a Display Packet by Name

To find a display packet of a specific name or beginning with a specific partial name, do the following:

1. From the Display Resources Tool Box, choose *Edit*.
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. In the Display Resource Editor form, choose *View – Find Packet/LP*.

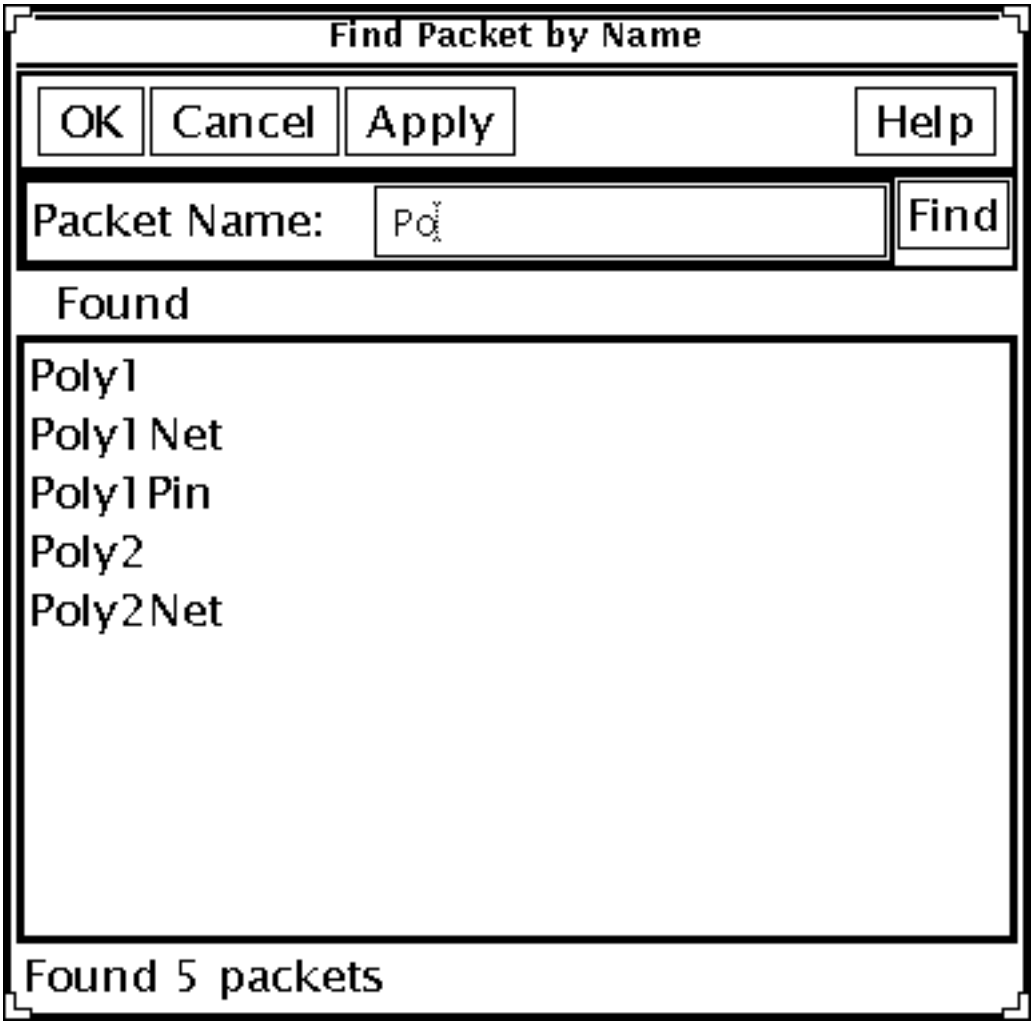
   The Display Resource Editor displays the Find Packet by Name form.



For a description of this form, see Appendix A.

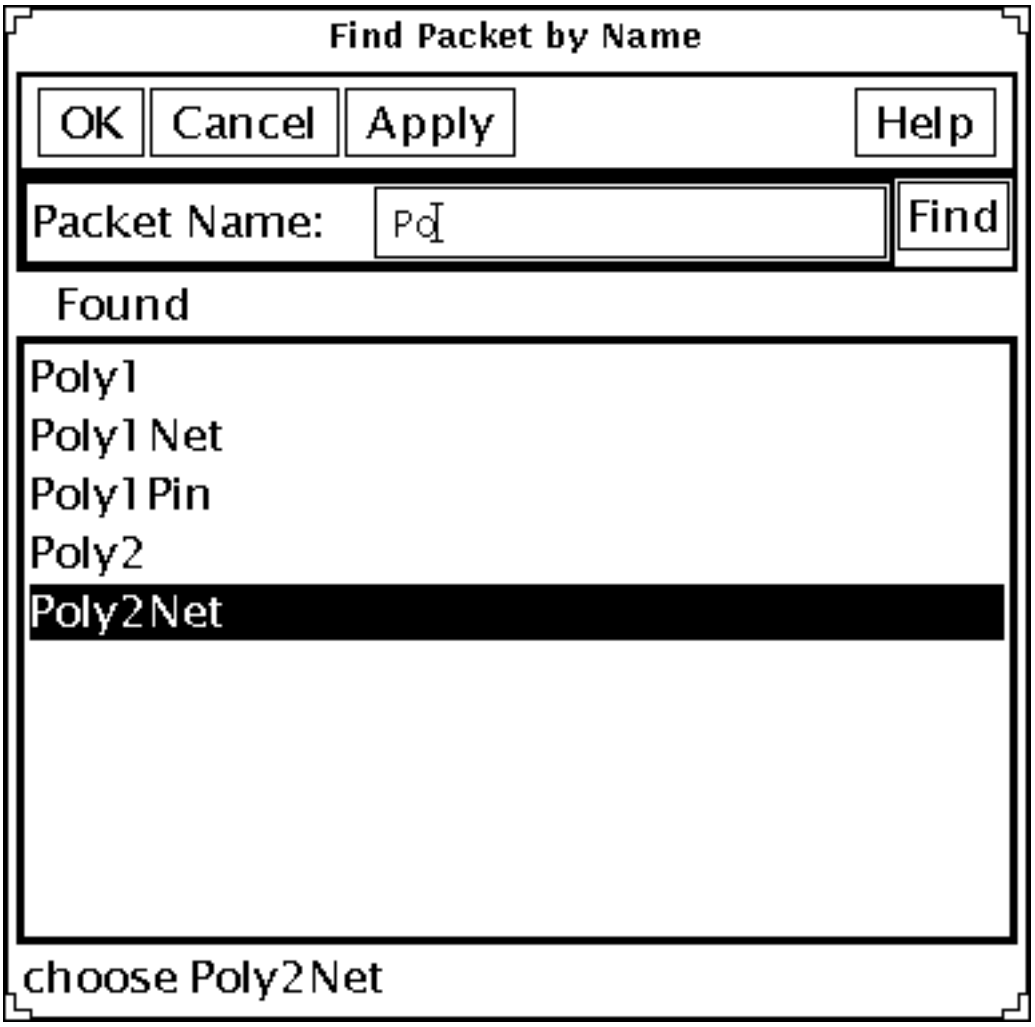
4. In the *Packet Name* field, type the name of the display packet you want to find.

As you type each character of the name, the Display Resource Editor displays, in the *Found* list box, a list of the names of all display packets that match what you have typed; it automatically updates the list of matching names with each subsequent character you type. For example:



5. In the *Found* list box, click on the name of the display packet you want to find.

The Display Resource Editor highlights the display packet name you selected.

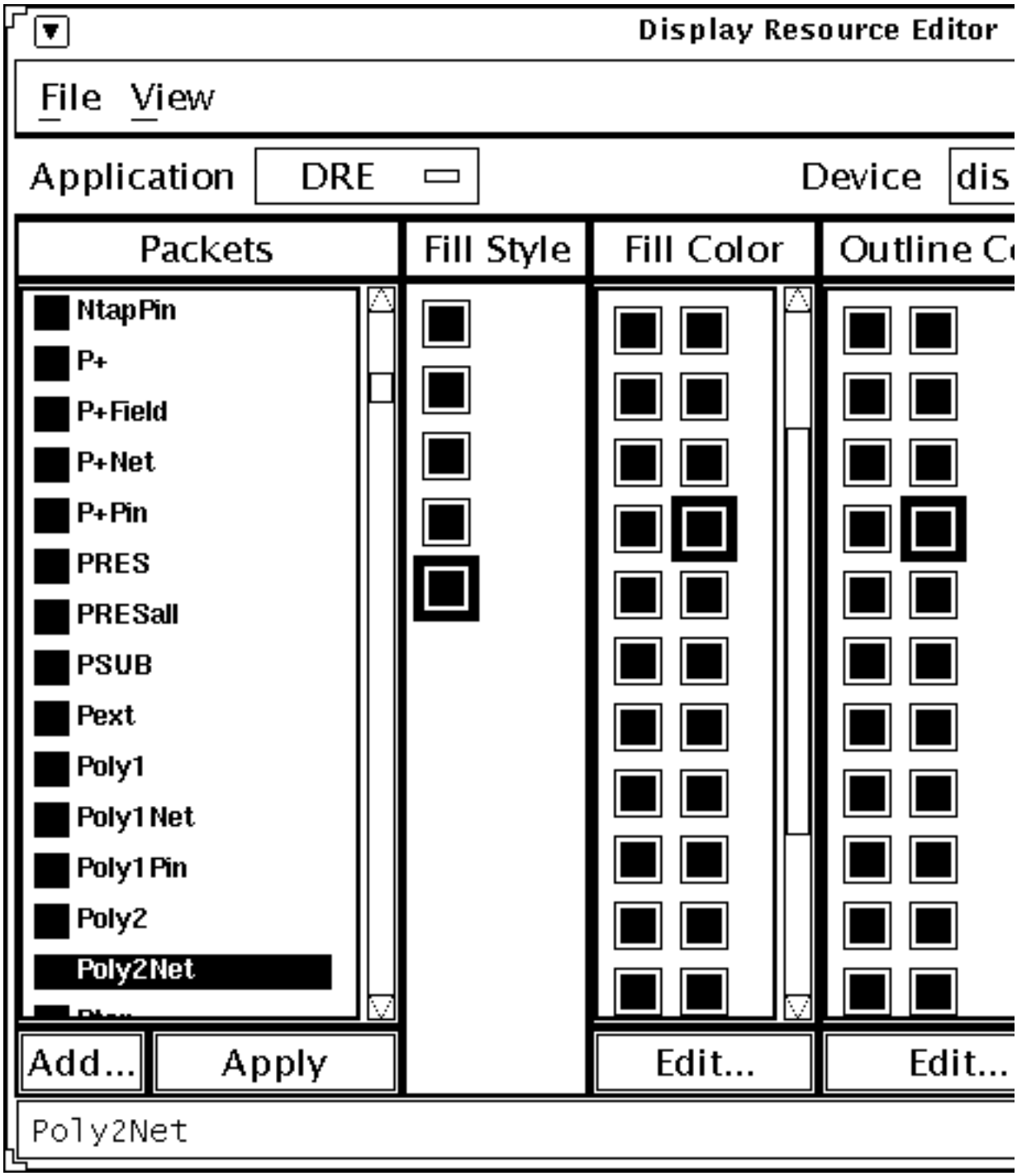


6. Click *Apply* or double-click on the name you are selecting if you want to keep the form to find more display packets; click *OK* if you are finished finding display packets.

When you click *Apply* or double-click on the selected name, the Display Resource Editor selects the display packet in its *Packets* list box and continues to display the Find Packet by Name form.

When you click *OK*, the Display Resource Editor selects the display packet in its *Packets* list box and dismisses the Find Packet by Name form.

You can then edit the packet definition as required. (See "Changing a Display Packet Definition" on page 312 for details.)



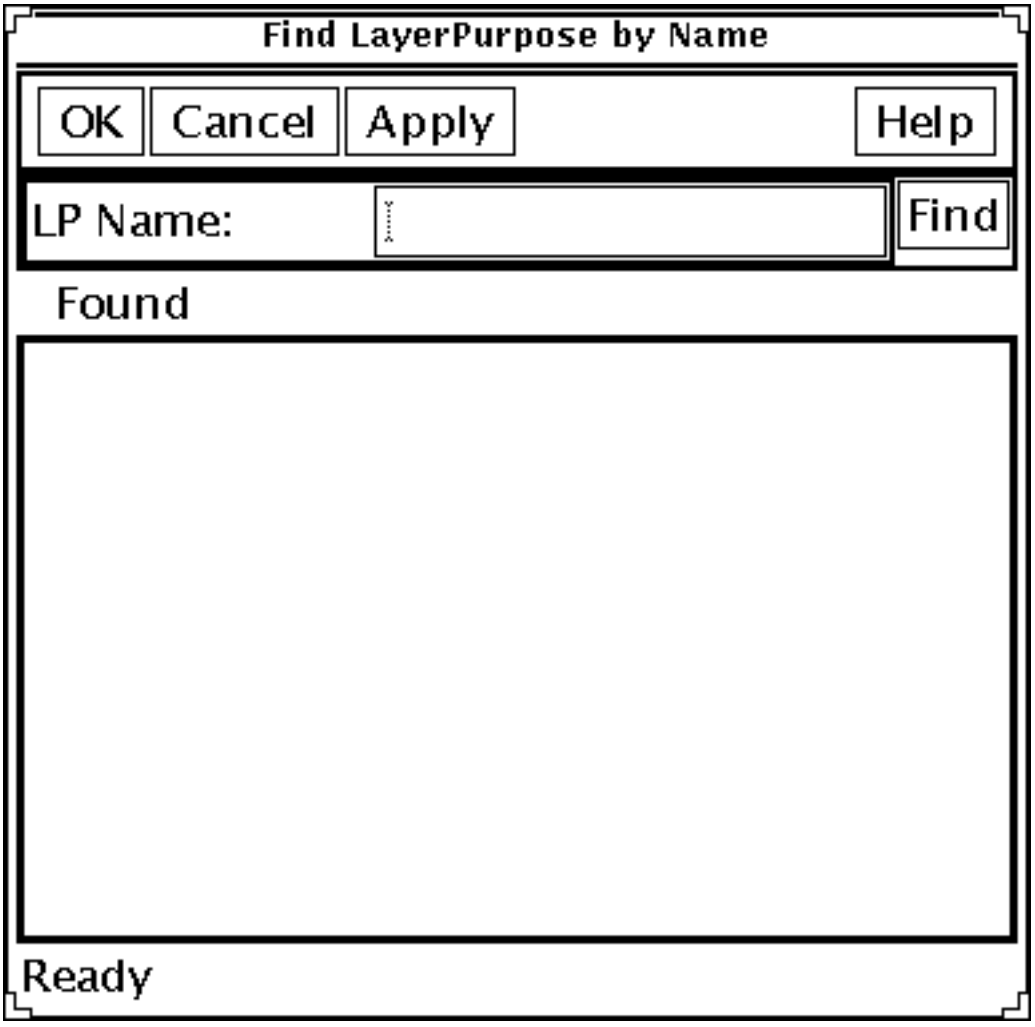
Finding a Layer-Purpose Pair by Name

To find a layer-purpose pair of a specific name or beginning with a specific partial name, do the following:

1. From the Display Resources Tool Box, choose *Edit*.
2. In the Display Resource Editor form, choose the application you use from the *Application* cyclic field.

   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. In the Display Resource Editor form, choose *View – Find Packet/LP*.

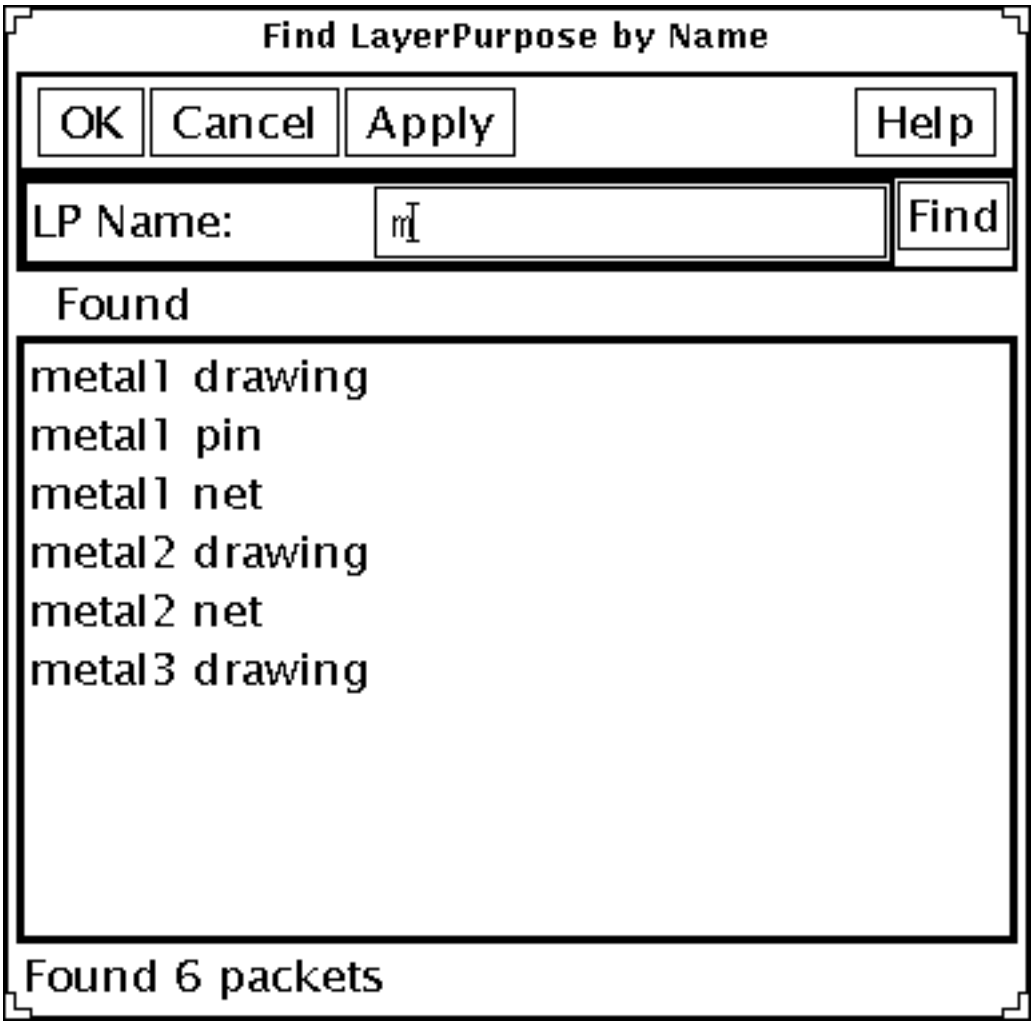
   The Display Resource Editor displays the Find LayerPurpose by Name form.



For a description of this form, see Appendix A.

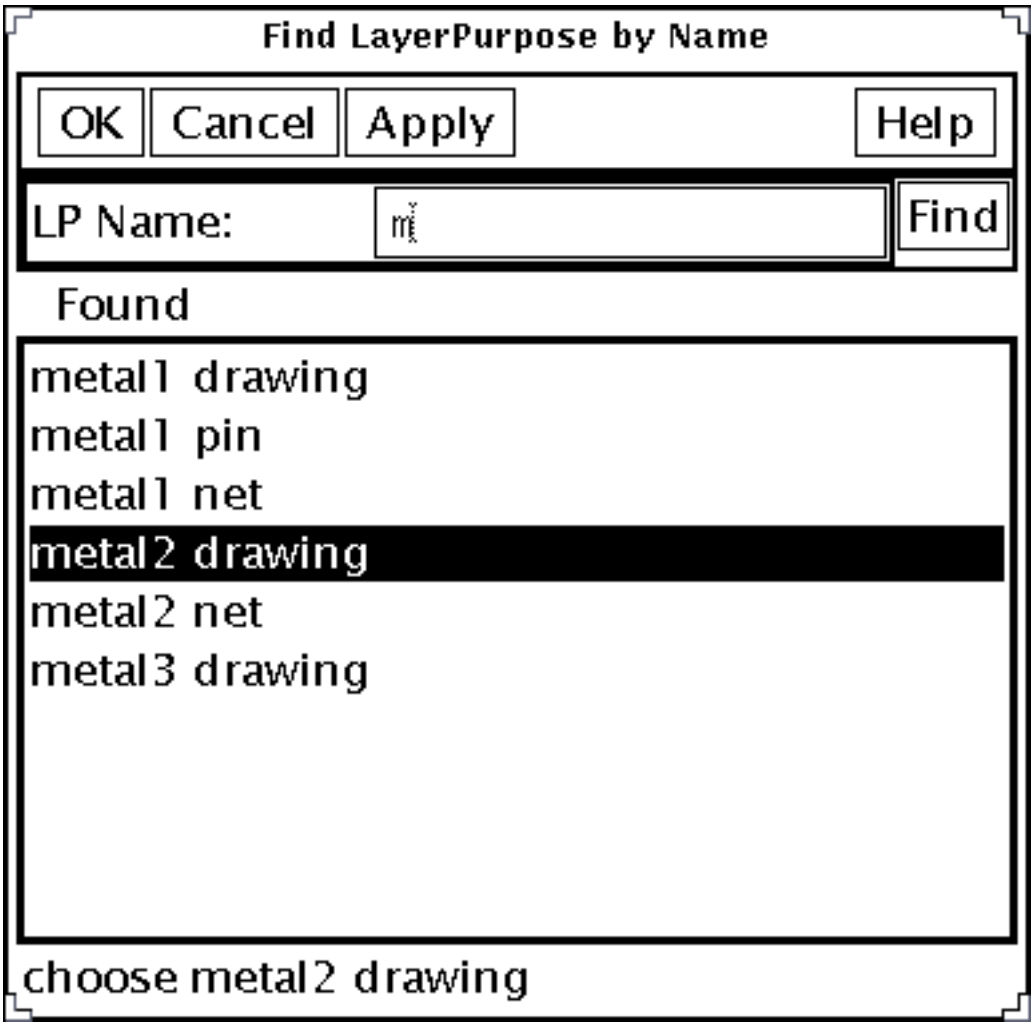
4. In the *LP Name* field, type the name of the layer-purpose pair you want to find.

As you type each character of the name, the Display Resource Editor displays, in the *Found* list box, a list of the names of all layer-purpose pairs that match what you have typed; it automatically updates the list of matching names with each subsequent character you type. For example:



5. In the *Found* list box, click on the name of the layer-purpose pair you want to find.

The Display Resource Editor highlights the layer-purpose pair name you selected.

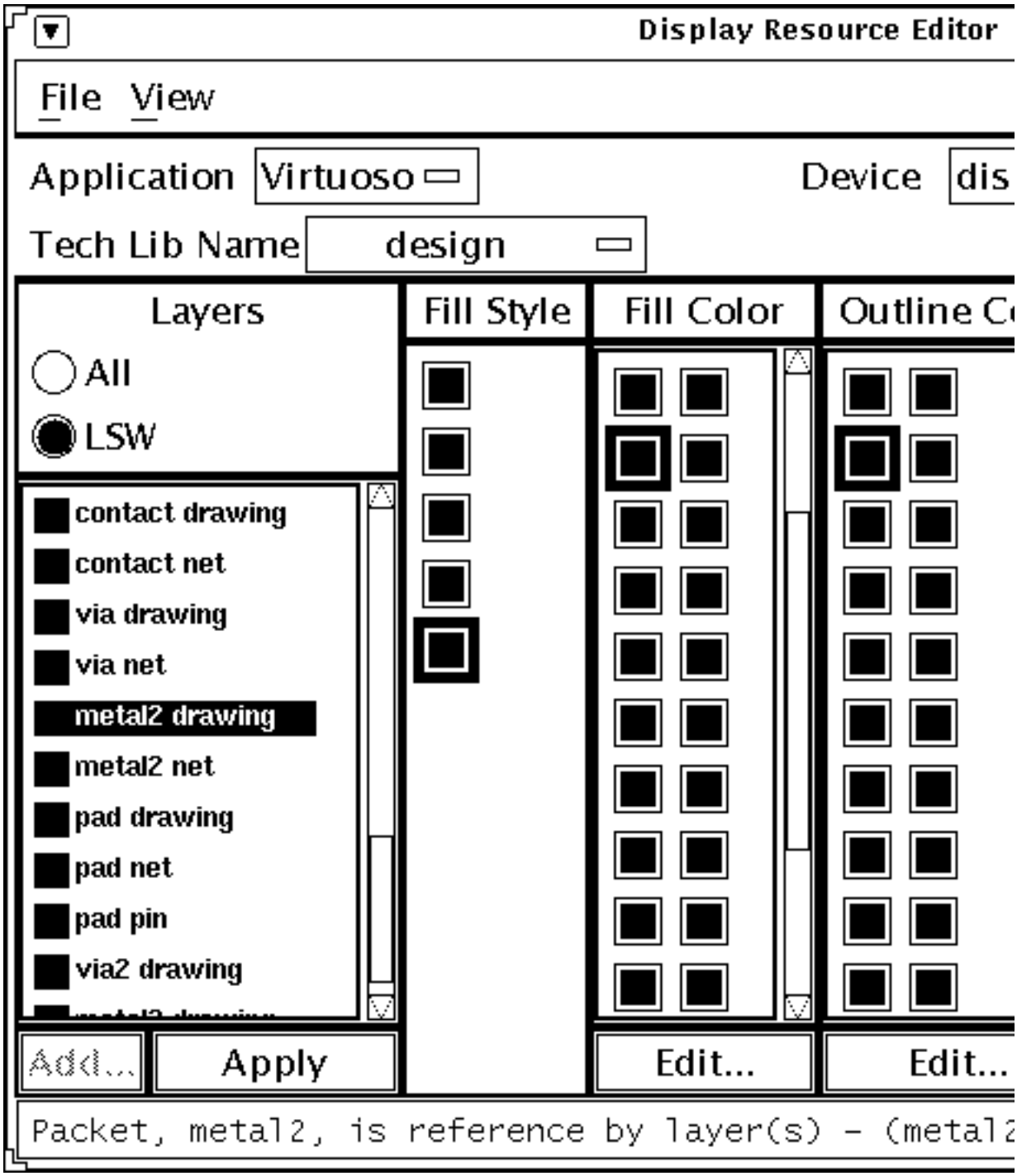


6. Click *Apply* or double-click on the name you are selecting if you want to keep the form to find more layer-purpose pairs; click *OK* if you are finished finding layer-purpose pairs.

When you click *Apply* or double-click on the selected name, the Display Resource Editor selects the layer-purpose pair in its *Layers* list box and continues to display the Find LayerPurpose by Name form.

When you click *OK*, the Display Resource Editor selects the layer-purpose pair in its *Layers* list box and dismisses the Find LayerPurpose by Name form.

You can then edit the display resources for the layer-purpose pair as required. (See "Changing a Layer Display" on page 313 for details.)



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 Display Resources Tool Box, choose *Edit*.
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. Do one of the following:

   ❑ Click on the name of the display packet you want to edit.

   ❑ Use the Find Packet by Name form to find and select a display packet as described in "Finding a Display Packet by Name" on page 304.

   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 Display Resources Tool Box, choose *Edit*.

   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. Do one of the following:

   - Click the layer you want to change.
   - Use the Find LayerPurpose by Name form to find and select a layer-purpose pair as described in "Finding a Layer-Purpose Pair by Name" on page 308.

   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. To scroll the message bar, click at the far right of the bar and press the right arrow on your keyboard.

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

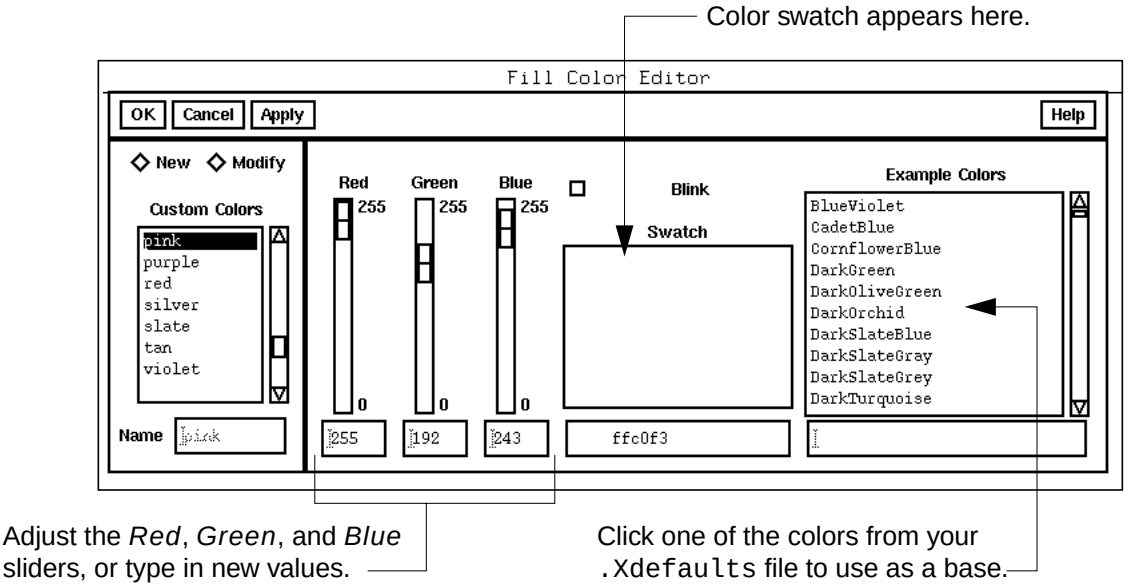
Fill and outline colors come from the same set of colors. To add a new color to the set, do the following:

1. From the Display Resources Tool Box, choose *Edit*.

   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.



For a description of this form, see Appendix A.

If a color does not appear in the *Swatch* area, refer to "Using Colormaps" in the *Design Framework II Configuration Guide*.

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. To make the color a blinking color, click the *Blink* button.
7. 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 Display Resources Tool Box, choose *Edit*.

   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.

   If a color does not appear in the *Swatch* area, refer to "Using Colormaps" in the *Design Framework II Configuration Information Manual*.
3. Click *Modify*.
4. Move the *Red*, *Green*, and *Blue* sliders, or type new values in the text fields below the sliders.
5. To make the color a blinking color, click the *Blink* button.
6. 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 Display Resources Tool Box, choose *Edit*.

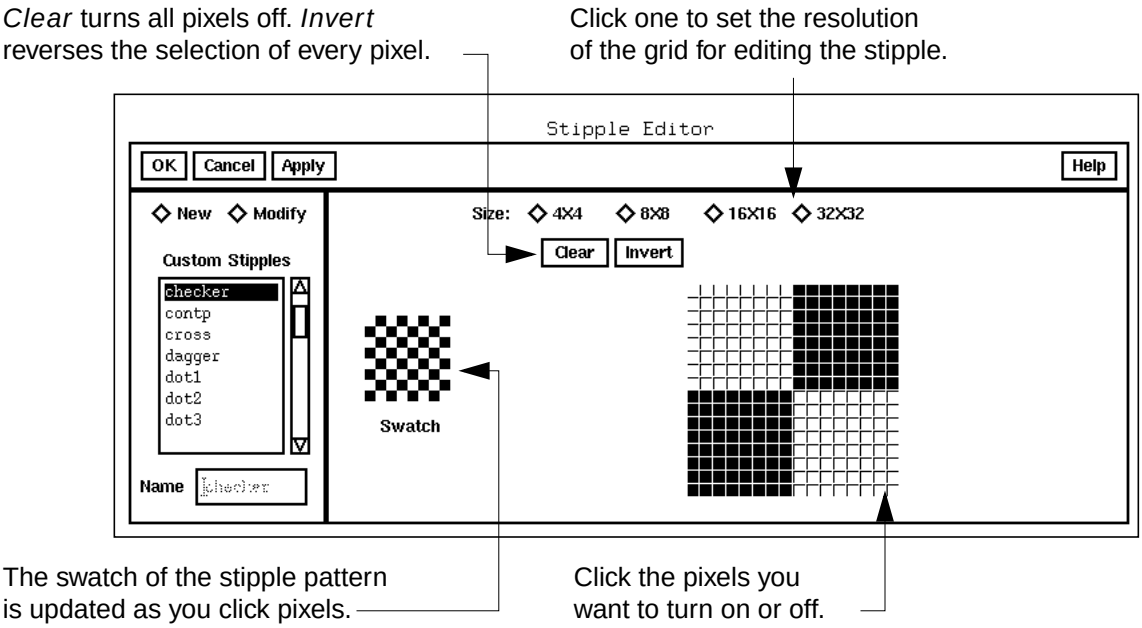
   The Display Resource Editor form appears.

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.



   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 Display Resources Tool Box, choose *Edit*.

   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:

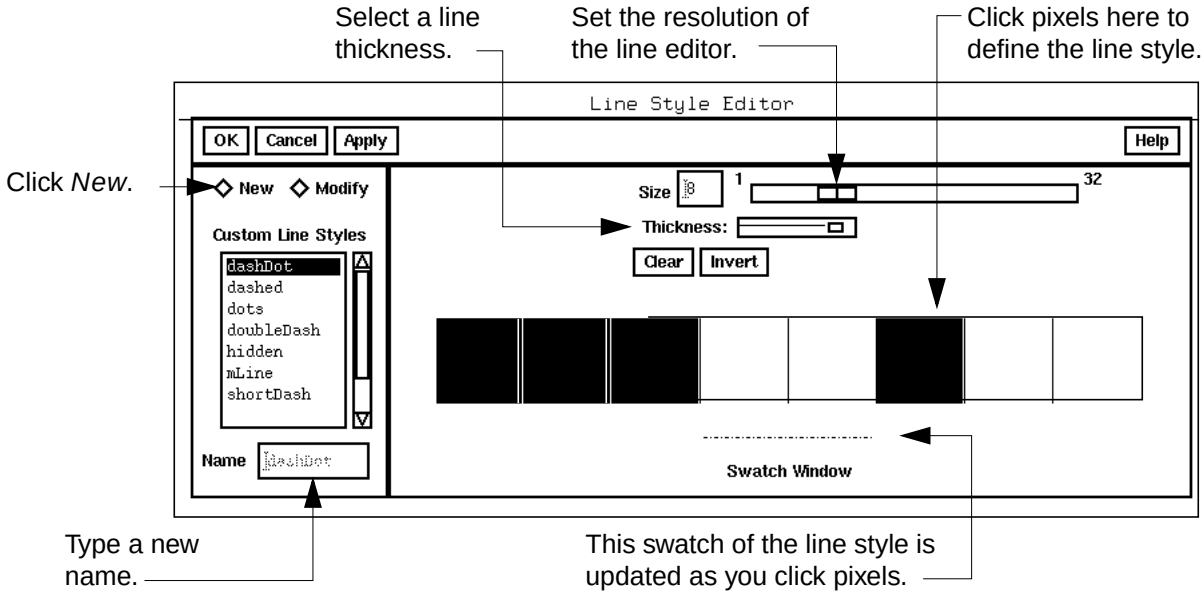
1. From the Display Resources Tool Box, choose *Edit*.

   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.



   For a description of this form, see Appendix A.

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 Display Resources Tool Box, choose *Edit*.

   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 Packet

To add a display packet, do the following:

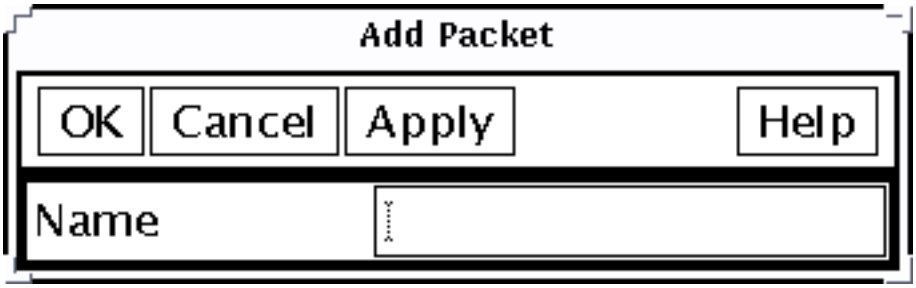
1. From the Display Resources Tool Box, choose *Edit*.
2. In the Display Resource Editor form, choose *DRE* from the *Application* cyclic field.

   The display packets defined for the software session appear in the left column of the form and the *Add* button is enabled.

3. On the Display Resource Editor form, click the display resources you want for the new display packet.

4. Click *Add*.

   The Display Resource Editor displays the Add Packet form.



   For a description of this form, see Appendix A.

5. In the *Name* field, type the name of the new display packet.

6. Click *Apply* if you want to add more display packet definitions or *OK* if you are finished adding display packet definitions.

   When you click *Apply*, the Display Resource Editor adds the defined display packet to its *Packet* list box and continues to display the Add Packet form.

   When you click *OK*, the Display Resource Editor adds the defined display packet to its *Packet* list box and dismisses the Add Packet form.

After adding a display packet, you can edit it as described in "Changing a Display Packet Definition" on page 312.

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 157.

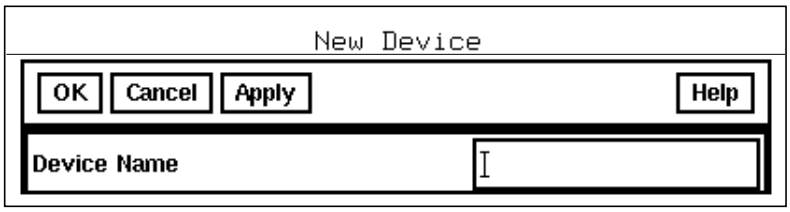
To add a display device, do the following:

1. From the Display Resources Tool Box, choose *Edit*.

   The Display Resource Editor form appears.

2. Click *New*.

   The New Device form appears.



   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:

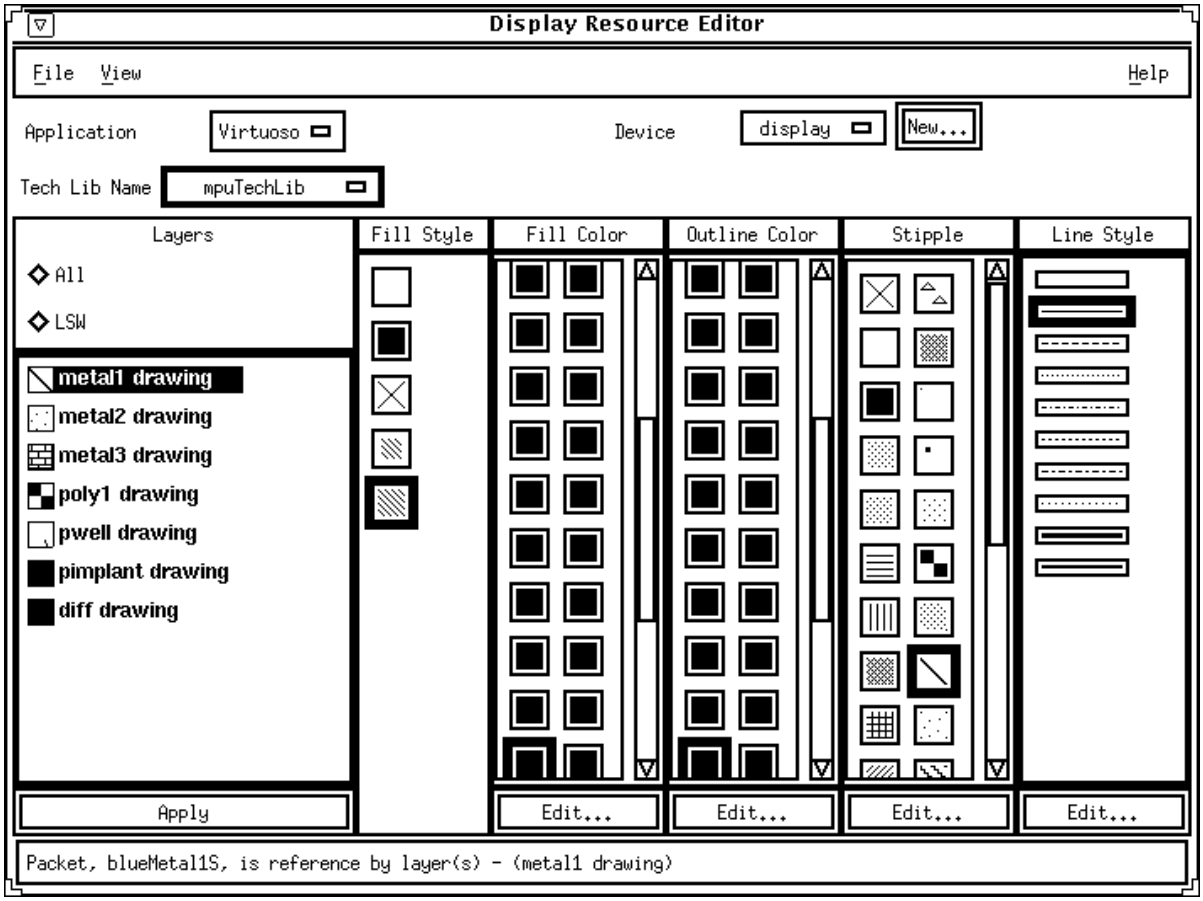
- Display packets are defined by a set of display resources. If you delete a display resource, display packets using the resource become invalid, as do all of the layers that use those display packets.
- To preserve the affected display packets, you must update the display packet definitions to include 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 Display Resources Tool Box, choose *Edit*.

   The Display Resource Editor appears.



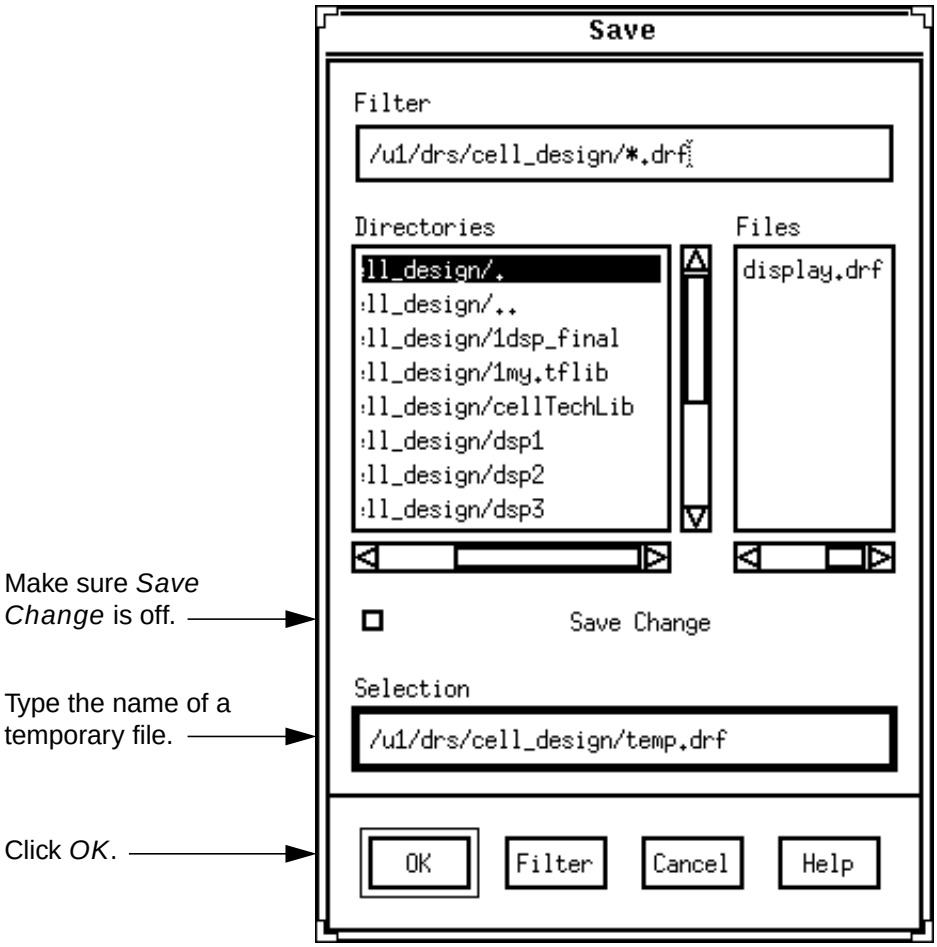
2. Choose *File – Save*.

   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:

```
;( 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 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 )
                             ) )
```

- 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 Cadence Design Software Handles Multiple Display Resource Files" on page 33.

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:

```
;( 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 Packets Affected by Deleting Display Resources" on page 322, 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

```
;( DisplayName PacketName     Stipple LineStyle Fill Outline)
 ( display     grayslashwide  slash   wide      gray gray)
```

     becomes this

```
;( 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:

```
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: With DFII only, you can also use SKILL functions to delete display resources in virtual memory. See "Editing Display Resource Data with SKILL Functions (Design Framework II Only)" on page 328 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 Display Resource Editor, select the File – Reinitialize command.

For information about where the source display resource files exist, refer to "How Cadence Design Software Handles Multiple Display Resource Files" on page 33.

Note: With DFII only, you can also use SKILL functions to delete display resources in virtual memory. See "Editing Display Resource Data with SKILL Functions (Design Framework II Only)" on page 328.

Editing Display Resource Data with SKILL Functions (Design Framework II Only)

The following table lists the SKILL functions that return information about and operate on display resource data:

SKILL Function	Description
drDeleteDisplay()	Deletes the specified display device from virtual memory.
drDeleteColor()	Deletes the definition of the specified color for the specified display device from virtual memory.
drDeleteLineStyle()	Deletes the specified line style from virtual memory.
drDeletePacket()	Deletes the definition of the specified display packet for the specified display device from virtual memory.
drDeleteStipple()	Deletes the definition of the specified stipple for the specified display device from virtual memory.
drFindPacket()	Reads virtual memory and returns a list of attributes of the specified display packet for the specified display device.
drGetColor()	Reads virtual memory and returns the display device; color name; and the red, green, blue, and blink values for the specified color.
drGetDisplay()	Reads virtual memory and returns the display device identifier of the specified display device name.
drGetDisplayIdList()	Reads virtual memory and returns a complete list of display device identifiers.
drGetDisplayName()	Reads virtual memory and returns the display device name of the specified display device identifier.
drGetDisplayNameList()	Reads virtual memory and returns a complete list of display device names.

SKILL Function	Description
drGetLineStyle()	Reads virtual memory and returns the display device name and the line style name, thickness, and pattern for the specified line style.
drGetLineStyleIndexByName()	Reads virtual memory and returns the line style index number for the specified line style for the specified display device.
drGetPacketList()	Reads virtual memory and returns a list of display packets that are defined for the specified display device.
drGetPacketAlias()	Reads virtual memory and returns a list of display packets that are aliased to the specified display packet.
drGetPacketFillStyle()	Reads virtual memory and returns the fill style number of the specified display packet for the specified display device.
drGetStipple()	Reads virtual memory and returns the display device name and the stipple name, width, height, and pattern for the specified stipple.
drGetStippleIndexByName()	Reads virtual memory and returns the stipple index number for the specified stipple name.
drLoadDrf()	Loads the display resource file (usually called `display.drf`) from any location.
drSetPacket()	Updates the value of the specified display packet for the specified display device in virtual memory.

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 Display Resources Tool Box, choose *Edit*.

   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, just as it does when you first start the software.

To update the cellview display, do the following:

➤ From the cellview menu, choose *Window – Redraw*.

For more information about how display resource data is loaded, refer to "How Cadence Design Software Handles Multiple Display Resource Files" on page 33.

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 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.

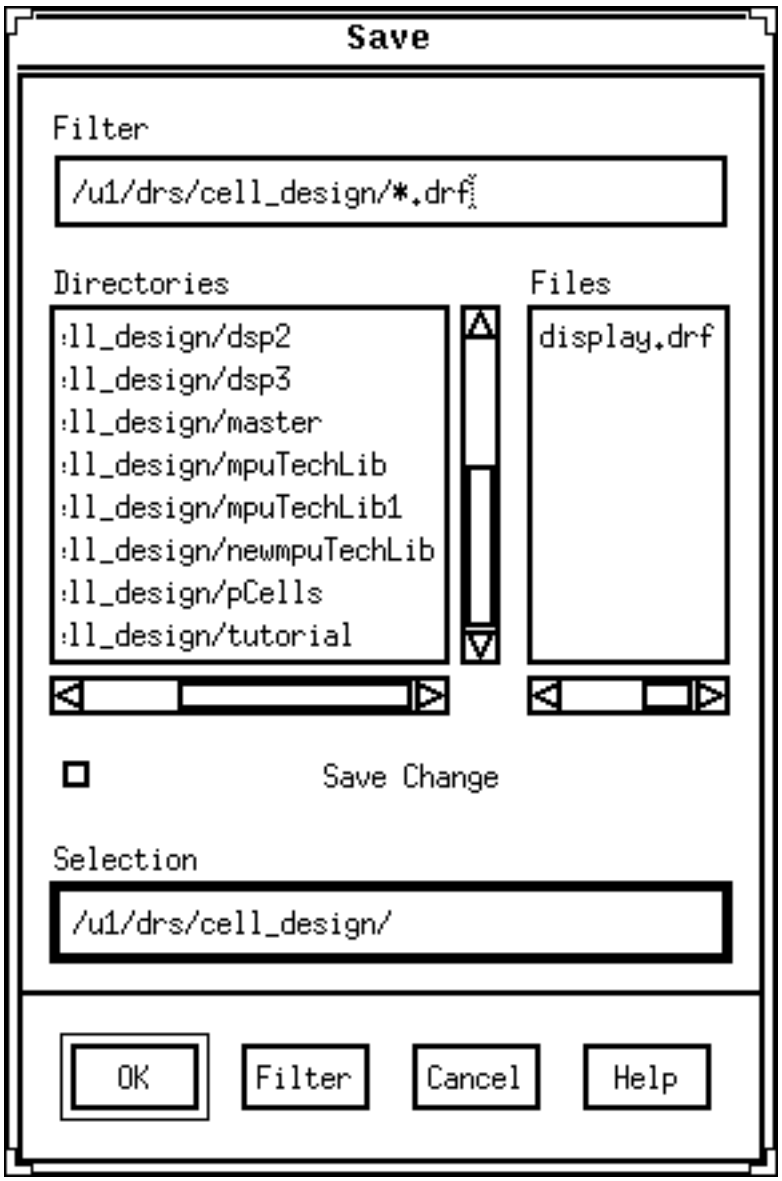
To save data loaded in the current session to a file, do the following:

1. From the Display Resources Tool Box, choose *Edit*.

   The Display Resource Editor form appears.

2. Choose *File – Save*.

   The Save form appears.



For a description of this form, see Appendix A.

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 Display Resources Tool Box, choose *Edit*.

   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 DFII CIW and in the `techManager.log` file.

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.

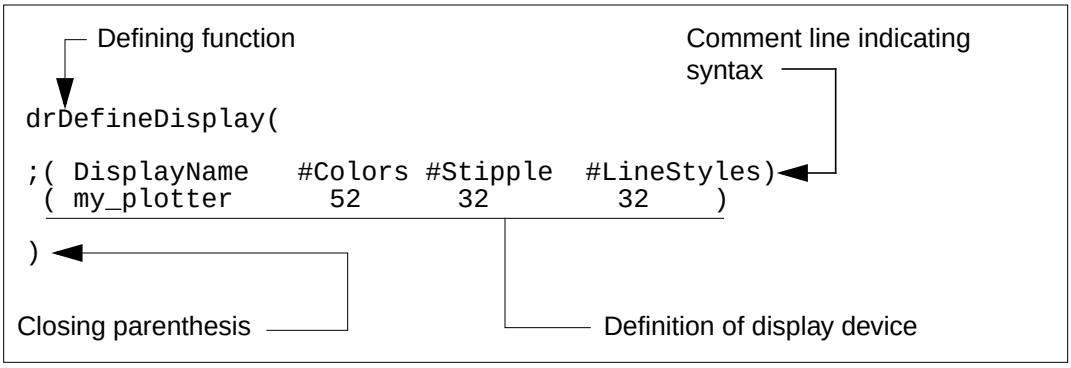
3. To keep a record of what the file contains, or why you have customized some display resources, insert comments in the file.

   For example:

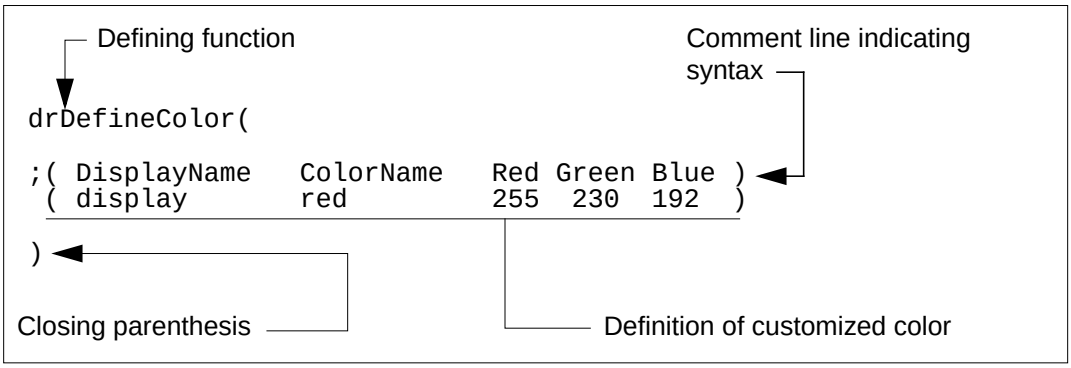
   ```
   ; Created 1/24/96 for Emerald project.
   ; Changed red color to look tan.
   ```

4. Delete all lines except those defining the display resources you have customized.

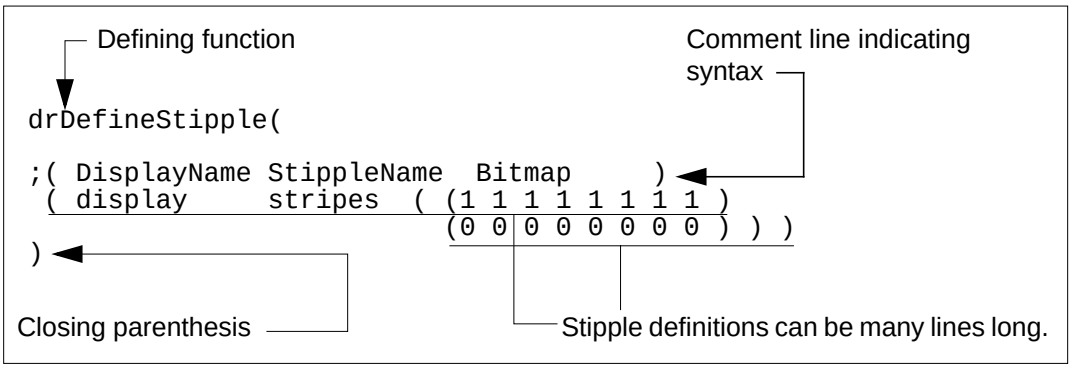
- To customize a display device, preserve the lines defining the display device and the colors, stipples, line styles, and display packets that use it. Here is an example of what to preserve for the display device itself:



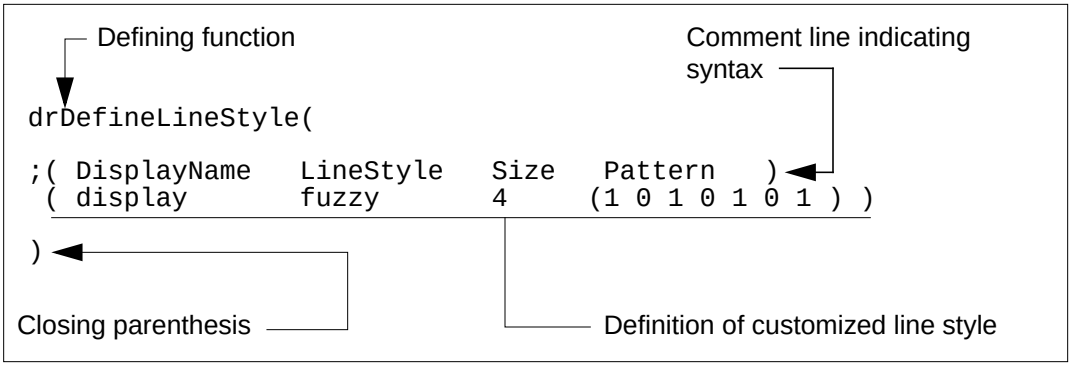
- To customize a color, preserve the lines shown in this example:



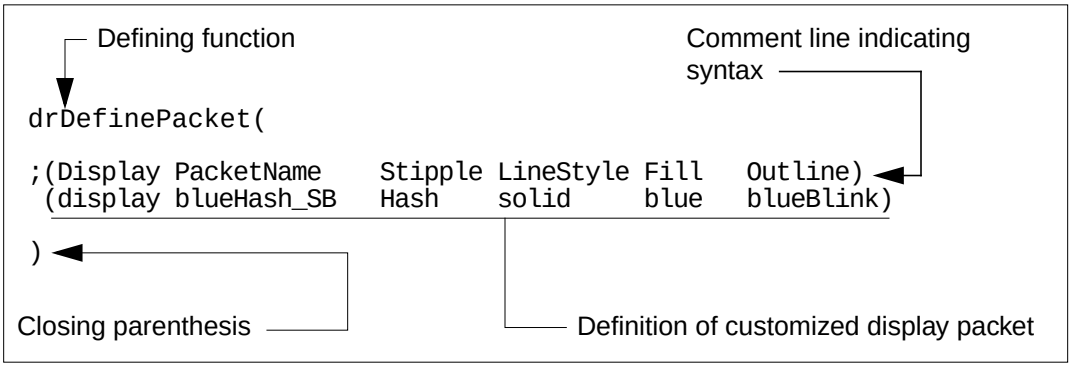
- To customize a stipple, preserve the lines shown in this example:



- To customize a line style, preserve the lines shown in this example:



- To customize a display packet, preserve the lines shown in this example:



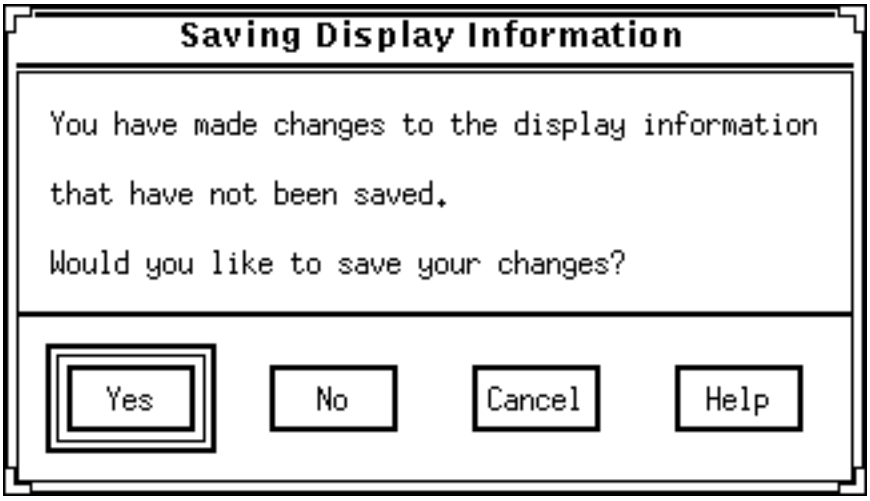
5. Save your changes and exit the text editor.
6. Test your customized resource data file.
7. When the file works as you expect, do one of the following:
   - Paste the customized resource definitions into one of the source display resource files
   - Place the file in your home directory or work area directory

   Your changes are automatically loaded every time you start the DFII software.

About Save Changes Message Boxes

Both the Display Resource Editor (and, with DFII, 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 dialog boxes discussed below appears.

The Saving Display Information dialog box appears if you try to exit the Display Resource Editor before saving your changes.



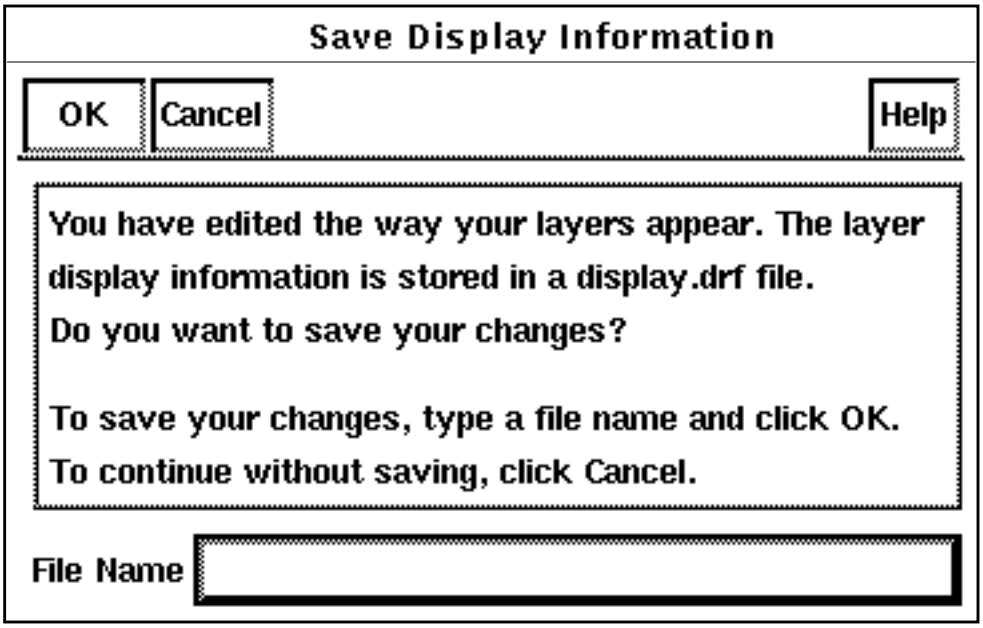
➤ To save your changes, click *Yes*.

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, in DFII, 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*.

A

Form Descriptions

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 (DFII Only)

```
tcDisplayNewTechForm()
```

This SKILL function displays the New Technology Library form. It is equivalent to the Technology File Tool Box *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 the individual applications 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 (DFII Only)

```
tcDisplayCheckTechForm()
```

This SKILL function displays the Check Technology File form. It is equivalent to the Technology File Tool Box *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 (DFII Only)

```
tcDisplayAttachTechForm()
```

This SKILL function displays the Attach Technology Library to Design Library form. It is equivalent to the Technology File Tool Box *Attach* 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.

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 (DFII Only)

```
tcDisplayDumpTechForm()
```

This SKILL function displays the Dump Technology File form. It is equivalent to the Technology File Tool Box *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 (DFII Only)

`tcDisplayLoadTechForm()`

This SKILL function displays the Load Technology File form. It is equivalent to the Technology File Tool Box *Load* command; it is also equivalent to the `tcDisplayCompTechForm()` SKILL function.

`tcDisplayCompTechForm()`

This SKILL function displays the Load Technology File form. It is equivalent to the Technology File Tool Box *Load* command; it is also equivalent to the `tcDisplayLoadTechForm()` SKILL function.

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.

Previous is enabled when there are display pages of layer-purpose pairs previous to the one currently displayed and lets you display the previous display page.

Next is enabled when there are more display pages of layer-purpose pairs ahead of the one currently displayed and lets you display the next display page. The Layer Purpose Pair Editor form displays a maximum of 50 layer-purpose pairs at a time; *Next* is enabled immediately when there are more than 50 layer-purpose pairs defined.

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 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 (DFII Only)

```
tcDisplaySetupTechForm()
```

This SKILL function displays the Technology File Set Up form. It is equivalent to the Technology File Tool Box *Edit Rules* 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 (DFII Only)

```
tcDisplayDiscardTechForm()
```

This SKILL function displays the Discard Edits To Technology File form. It is equivalent to the Technology File Tool Box *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 (DFII Only)

```
tcDisplaySaveTechForm()
```

This SKILL function displays the Save Technology File form. It is equivalent to the Technology File Tool Box *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.

Abbrv. 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.

Abbrv. 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 the Diva® verification products 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.

Abbrv. 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.

Abbrv. 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 the Diva verification products 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 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.

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 filename
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.

Layer Functions

Layer displays the name of the layer in the layer functions rule currently selected and lets you type a layer name.

Function displays the function currently assigned to the layer and lets you choose a function to assign to 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`.

1 Layer Table lets you select `tableSpacingRules` for one-layer tables.

2 Layer Table lets you select `tableSpacingRules` for two-layer tables.

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*, *1 Layer Table*, *2 Layer Table*, 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*.

Remove 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:

Rule displays the name of the rule currently selected and lets you type a rule name.

Edit table brings up the Technology File – Edit Physical Rules Table form.

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` or `tableSpacingRules` that specify

one layer; it applies to `spacingRules` and `tableSpacingRules` 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 – Edit Physical Rules Table Form

Table name is the name of the physical rules table currently selected for editing. You cannot change this field on this form; you must edit the table name on the Technology File – Physical Rules form.

Table Dimension lets you select whether the table is one- or two-dimensional.

Index1 lets you specify the index name for the index1 entries in the first field. The second and third fields are for specifying the index and match type for a table entry for index1.

Index2 lets you specify the index name for the index2 entries in the first field. The second and third fields are for specifying the index and match type for a table entry for index2.

Value Type lets you select the value type for the value associated with the index or indices.

Value lets you type the value associated with the index or indices for the table entry.

Table Entry list box initially displays the table entries. As you edit, delete, and add data, this list box displays the changed data.

Up lets you highlight a table entry and move it up one position in the table.

Down lets you highlight a table entry and move it down one position in the table.

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*.

Remove deletes the table entry currently selected in the *Table Entry* list box.

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`.

1 Layer Table lets you select `tableCharacterizationRules` for one-layer tables.

2 Layer Table lets you select `tableCharacterizationRules` for two-layer tables.

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*, *1 Layer Table*, *2 Layer Table*, 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*.

Remove 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:

Rule displays the name of the rule currently selected and lets you type a rule name.

Edit table brings up the Technology File – Edit Electrical Rules Table form.

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` or `tableCharacterizationRules` that specify one layer; it applies to `characterizationRules` or `tableCharacterizationRules` 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 – Edit Electrical Rules Table Form

Table name is the name of the electrical rules table currently selected for editing. You cannot change this field on this form; you must edit the table name on the Technology File – Electrical Rules form.

Table Dimension lets you select whether the table is one- or two-dimensional.

Index1 lets you specify the index name for the index1 entries in the first field. The second and third fields are for specifying the index and match type for a table entry for index1.

Index2 lets you specify the index name for the index2 entries in the first field. The second and third fields are for specifying the index and match type for a table entry for index2.

Value Type lets you select the value type for the value associated with the index or indices.

Value lets you type the value associated with the index or indices for the table entry.

Table Entry list box initially displays the table entries. As you edit, delete, and add data, this list box displays the changed data.

Up lets you highlight a table entry and move it up one position in the table.

Down lets you highlight a table entry and move it down one position in the table.

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*.

Remove deletes the table entry currently selected in the *Table Entry* list box.

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 `leLswLayers` 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 – 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 Compactor rule and allows you to type data. The fields displayed are as follows for each `compactorRules` 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 – PR Rules Form

Technology Library is the library that contains the technology file currently selected for editing. You cannot change the technology library on this form; you must change it on the Technology File Set Up form.

Subclass lets you select the subclass, either `prRoutingLayers`, `prViaTypes`, `prStackVias`, `prMastersliceLayers`, `prViaRules`, `prGenViaRules`, or `prNonDefaultRules`, for editing Place and Route rules.

Existing Rules list box initially displays the data currently in the technology file that corresponds to the *PR 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*.

Remove deletes the rule currently selected in the *Existing Rules* list box.

The bottom section of the form displays the data for the currently selected PR rule and allows you to type data. On the Technology File – PR 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 `prRules` subclass:

- prRoutingLayers

  Layer displays the name of the layer in the rule currently selected and lets you type a layer name.

  Direction displays the direction in the rule currently selected and lets you select a direction.

  Pitch displays the pitch in the rule currently selected and lets you type a pitch.This setting corresponds to the `prRoutingPitch` subclass.

  Offset displays the offset in the rule currently selected and lets you type an offset. This setting corresponds to the `prRoutingOffset` subclass.

- prViaTypes

  Cell displays the name of the cell in the rule currently selected and lets you type a cell name.

  View displays the name of the view in the rule currently selected and lets you type a view name.

Type displays the via type in the rule currently selected and lets you select a type.

- prStackVias

  Via Layer1 displays the name of the first via layer in the rule currently selected and lets you choose a via layer.

  Via Layer2 displays the name of the second via layer in the rule currently selected and lets you choose a via layer.

- prMastersliceLayers

  Layer displays the name of the layer in the rule currently selected and lets you type a layer name.

- prViaRules

  Rule displays the name of the rule currently selected and lets you type a rule name.

  Via Devices list box displays the names of the via devices in the rule currently selected.

  Up lets you highlight a via device and move it up one position in the list box.

  Down lets you highlight a via device and move it down one position in the list box.

  Edit lets you add a new via device name to the Via Devices list box after typing it into the field to the left of Edit.

  Remove lets you delete a highlighted via device from the list box.

  Top lets you select the layer and direction, type the minimum and maximum wire width, and type the minimum spacing between the contact cut and the outer edge of the via and the minimum overhang of the via to the wire for the top via layer.

  Bottom lets you select the layer and direction, type the minimum and maximum wire width, and type the minimum spacing between the contact cut and the outer edge of the via and the minimum overhang of the via to the wire for the bottom via layer.

- prGenViaRules

  Rule displays the name of the rule currently selected and lets you type a rule name.

  Layer displays the name of the layer in the rule currently selected and lets you type a layer name.

  Point Lower, Upper lets you type the lower and upper points of the via layer.

  Pitch X, Y lets you type the pitch.

  Res lets you type the resistance.

Top lets you select the layer and direction, type the minimum and maximum wire width, and type the minimum spacing between the contact cut and the outer edge of the via and the minimum overhang of the via to the wire for the top via layer.

Bottom lets you select the layer and direction, type the minimum and maximum wire width, and type the minimum spacing between the contact cut and the outer edge of the via and the minimum overhang of the via to the wire for the bottom via layer.

- prNonDefaultRules

Rule displays the name of the rule currently selected and lets you type a rule name.

Routing Layers list box displays the routing layers for the rule currently selected and lets you add, edit, or remove routing layer data.

Layer displays the name of the layer in the rule currently selected and lets you choose a layer name.

Width displays the width of the layer in the rule currently selected and lets you type a layer width.

Spacing displays the minimum spacing allowed for the layer in the rule currently selected and lets you type a minimum space.

Notch displays the notch spacing allowed for the layer in the rule currently selected and lets you type a notch spacing value.

Wire Ext displays the wire extension allowed for the layer in the rule currently selected and lets you type a wire extension.

Cap displays the capacitance for the layer in the rule currently selected and lets you type a capacitance.

Res displays the resistance for the layer in the rule currently selected and lets you type a resistance.

Edge Cap displays the edge capacitance for the layer in the rule currently selected and lets you type an edge capacitance.

Up lets you highlight a routing layer line in the list box and move it up one position in the list box.

Down lets you highlight a routing layer line in the list box and move it down one position in the list box.

Edit lets you add new routing layer data to the Routing Layers list box after entering it into the fields below the list box.

Remove lets you delete a highlighted routing layers line from the list box.

Vias list box displays the vias for the rule currently selected and lets you add, edit, or remove vias.

Via lets you type a via name to add to the list box.

Edit lets you add a new via to the Vias list box after entering it into the Via field below the list box.

Up lets you highlight a via in the list box and move it up one position in the list box.

Down lets you highlight a via in the list box and move it down one position in the list box.

Edit lets you add a new via to the Vias list box after typing it in the field below the list box.

Remove lets you delete a highlighted via from the list box.

Via Layers list box displays the via layers for the rule currently selected and lets you add, edit, or remove via layer data.

Via Layer displays the name of the first via layer in the rule currently selected and lets you choose a via layer.

Via Layer displays the name of the second via layer in the rule currently selected and lets you choose a via layer.

Spacing displays the minimum spacing allowed between the via layers in the currently selected rule and lets you type a minimum spacing value.

Stackable shows whether the via layers in the currently selected rule are stackable and lets you select stackable or turn stackable off.

Edit lets you add a new via layers line to the Via Layers list box after entering the data into the fields below the list box.

Remove lets you delete a highlighted via layers line from the list box.

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 that 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

File lets you load a display resource file, save changes to a display resource file to disk, reinitialize, or exit the Display Resource Editor.

View lets you turn labels on or off or find a display packet or layer-purpose pair.

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.

Add lets you add new display packets.

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.

SKILL Function to Display Form (DFII Only)

```
dreInvokeDre()
```

This SKILL function displays the Display Resource Editor form. It is equivalent to the Display Resources Tool Box *Edit* command.

Find Packet by Name Form
Find LayerPurpose by Name Form

Packet Name or **LP Name** lets you specify the name of the display packet or layer-purpose pair to find.

Found displays the display packets or layer-purpose pairs found matching the name or partial name typed into the *Name* field and lets you select the one you want to find in the Display Resource Editor form.

Apply selects the display packet or layer-purpose pair in the Display Resource Editor form and continues to display the Find Packet by Name form or the Find LayerPurpose by Name form.

OK selects the display packet or layer-purpose pair in the Display Resource Editor form and dismisses the Find Packet by Name form or the Find LayerPurpose by Name form.

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.

Add Packet Form

Name lets you specify the name of the new display packet to add.

Apply adds the new display packet, which is displayed in the *Packets* list box in the Display Resource Editor form, and continues to display the Add Packet form.

OK adds the new display packet, which is displayed in the *Packets* list box in the Display Resource Editor form, and dismisses the Add Packet form.

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.

B

System-Reserved Layers and Purposes

This appendix lists the

- System-reserved layers
- System-reserved purposes

Layers

The following table shows system-reserved layers.

Layer Name	Layer Purposes	Layer Number	Notes
Unrouted	drawing	200	
Unrouted	drawing1	200	
Unrouted	drawing2	200	
Unrouted	drawing3	200	
Unrouted	drawing4	200	
Unrouted	drawing5	200	
Unrouted	drawing6	200	
Unrouted	drawing7	200	
Unrouted	drawing8	200	
Unrouted	drawing9	200	
Row	drawing	201	Used for Preview Silicon Ensemble™ rows
Row	label	201	
Group	drawing, label	202	Used for floorplan/ placement group; applied to all place-and-route tools
Cannotoccupy	boundary, drawing	203	Used for Preview Gate Ensemble®
Canplace	drawing	204	Used for Preview Gate Ensemble
hardFence	drawing	205	
softFence	drawing	206	
y0	drawing	207	
y1	drawing	208	
y2	drawing	209	
y3	drawing	210	

Layer Name	Layer Purposes	Layer Number	Notes
y4	drawing	211	
y5	drawing	212	
y6	drawing	213	
y7	drawing	214	
y8	drawing	215	
y9	drawing	216	
designFlow	drawing, drawing1–9	217	
stretch	drawing	218	
edgeLayer	drawing, pin	219	
changedLayer	tool0, tool1	220	
unset	drawing	221	
unknown	drawing	222	
spike	drawing	223	
hiz	drawing	224	
resist	drawing	225	
drive	drawing	226	
supply	drawing	227	
wire	drawing, flight, label	228	
pin	annotate, drawing, label	229	
text	drawing, drawing1–2	230	
device	annotate, drawing, drawing1–2, label	231	
border	drawing	232	
snap	drawing	233	

Layer Name	Layer Purposes	Layer Number	Notes
align	drawing	234	
prBoundary	boundary, drawing, label	235	
instance	drawing, label	236	
annotate	drawing, drawing1–9	237	
marker	error, warning	238	
select	drawing	239	
winActiveBanner	drawing	240	
winAttentionText	drawing	241	
winBackground	drawing	242	
winBorder	drawing	243	
winBottomShadow	drawing	244	
winButton	drawing	245	
winError	drawing	246	
winForeground	drawing	247	
winInactiveBanner	drawing	248	
winText	drawing	249	
winTopShadow	drawing	250	
grid	drawing, drawing1	251	
axis	drawing	252	
hilite	drawing, drawing1–9	253	
background	drawing	254	

Purposes

The following table lists system-reserved purposes.

Layer Purpose	Purpose Number	Type of Data
`all`	255	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)
`annotate`	240	Schematic text
`boundary`	250	Boundary layer purpose you assign to layers of a place-and-route design (place-and-route tools)
`cell`	254	Not used by the Cadence® Design Framework (DFII) tools
`drawing`	252	Default layer purpose for drawing layers
`drawing1`	241	Purpose used by the system-reserved layers
`drawing2`	242	Purpose used by the system-reserved layers
`drawing3`	243	Purpose used by the system-reserved layers
`drawing4`	244	Used by the Interactive Wire Editor
`drawing5`	245	Purpose used by the system-reserved layers
`drawing6`	246	Purpose used by the system-reserved layers
`drawing7`	247	Purpose used by the system-reserved layers
`drawing8`	248	Purpose used by the system-reserved layers
`drawing9`	249	Purpose used by the system-reserved layers
`error`	239	Error markers for rule checking
`flight`	238	Flightlines in a schematic cellview
`label`	237	Layers for text data
`net`	253	Translated interconnect layers, Diva® verification products extracted views
`pin`	251	Pin layers
`tool0`	236	Markers for Diva verification products incremental DRC
`tool1`	235	Markers for Diva verification products incremental DRC

Layer Purpose	Purpose Number	Type of Data
warning	234	Warning markers for rule checking

C

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 394.

- The layers have conflicting layer numbers.

  Refer to “Resolving Inconsistent Layer Names” on page 400.

- The layer appearance is defined differently.

  Refer to “Fixing Layers That Do Not Appear Correctly” on page 401.

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/tools/dfII/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.

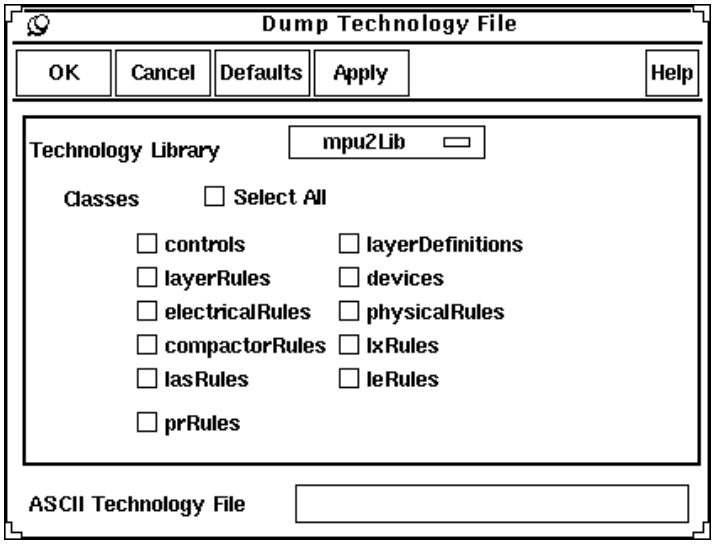
Adding a Layer by Manually Editing the Technology File

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 Technology File Tool Box, choose *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.

Adds the F+ layer →

```
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.

Adds the F+ layer purpose →

```
techLayerPurposePriorities(
 ;layers are ordered from lowest to highest priority
 ;( 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:

Packet	gold	Part of the Change Layer	no
Visibility	yes	Drag Enable	yes
Selectability	yes	Valid	yes

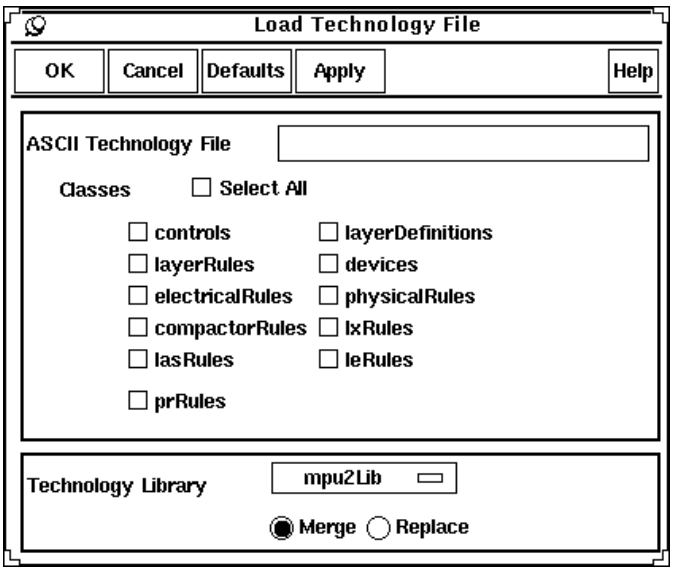
Sets the F+ layer attributes

```
techDisplays(
 ;( LayerName  Purpose  Packet   Vis Sel Con2ChgLy DrgEnbl Valid )
 ;( ---------  -------  ------   --- --- --------- ------- ----- )
  ( isolation drawing  cyan     t    t   t         t       t     )
  ( F+        drawing  gold     t    t   nil       t       t     )
  ( poly2     drawing  forest   t    t   t         t       t     )
  ( nwell     net      yellow   t    t   nil       t       nil   )
```

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 402.

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 Technology File Tool Box, *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.

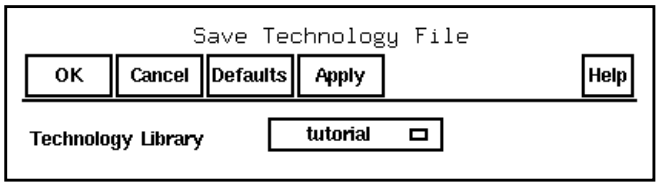
   c. From the *Technology Library* cyclic field, choose the technology library for this technology file.

   d. Set *Merge* on.

   e. Click *OK*.

9. Save these changes to disk by choosing, from the Technology File Tool Box, *Save*.

   The Save Technology File form appears.



   a. Set the *Technology Library* cyclic field to the technology library you want to save.

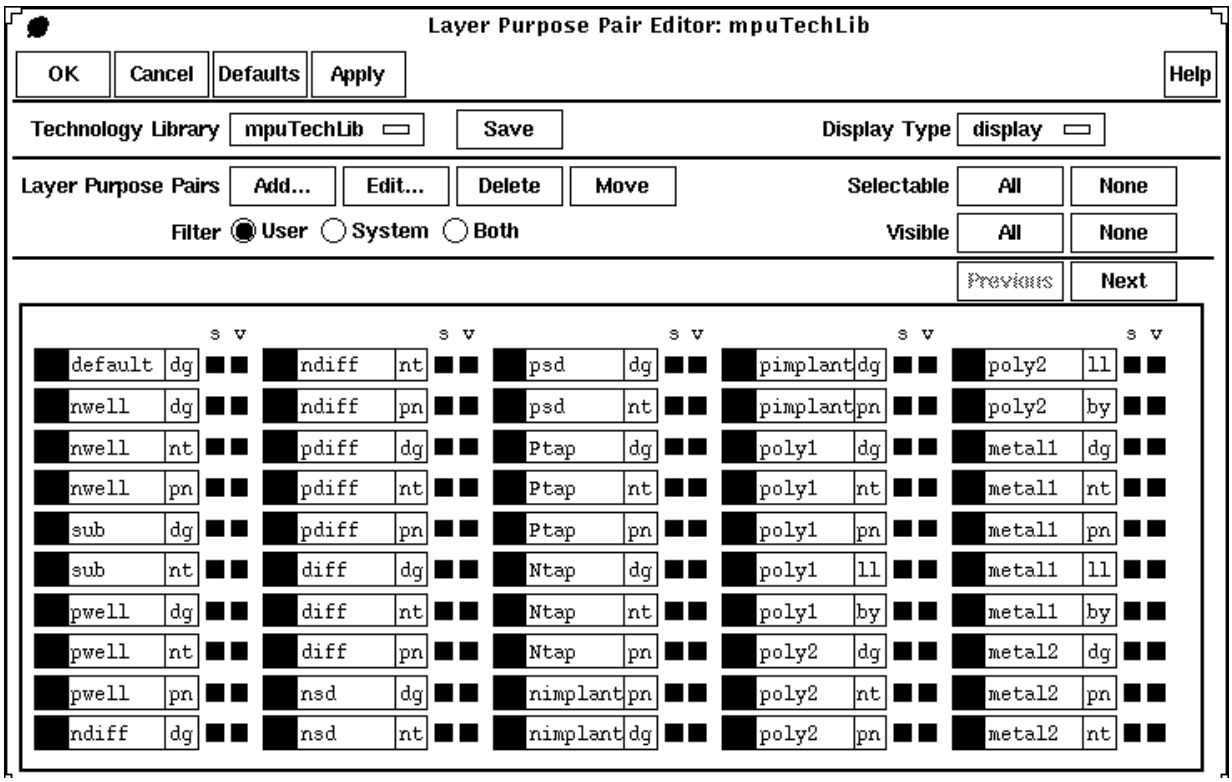
   b. Click *OK*.

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 Technology File Tool Box, choose *Edit Layers*.

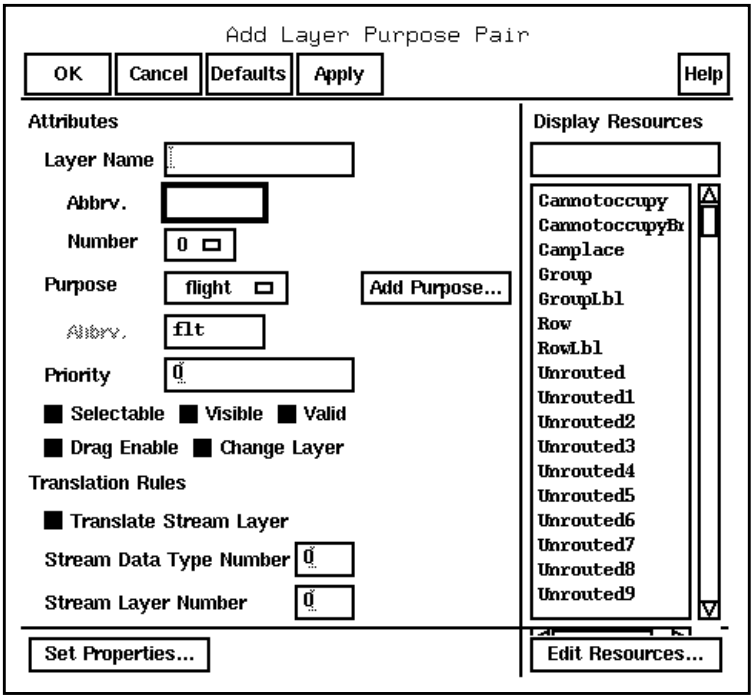
The Layer Purpose Pair Editor form appears.



2. Set the *Technology Library* cyclic field to the technology library you want to edit.
3. Check the form for missing layers.

   Note: If the Next button is enabled, be sure to check all layer-purpose pairs. The Layer Purpose Pair Editor displays a maximum of 50 layer-purpose pairs at a time. When there are more layer-purpose pairs, the *Next* button is enabled. Click *Next* to display the next page of layer-purpose pairs to choose from. The *Previous* button is enabled when you page forward through layer-purpose pairs. When the *Next* button is disabled, there are no further pages of layer-purpose pairs ahead of the currently displayed list; when the *Previous* button is disabled, there are no further pages of layer-purpose pairs behind the currently displayed list.
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 Technology File Tool Box, *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 Technology File Tool Box, choose *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

```
techLayers(
 ;( LayerName        Layer#        Abbreviation  )
 ;( ---------        ------        ------------  )
 ;User-Defined Layers:
  ( pdiff             2             pdiff         )
  ( nwell             9             nwell         )
  ( pwell             6             pwell         ) )
```

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 Technology File Tool Box, choose *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*.
   g. Save these changes to disk by choosing *Technology File – Save*.

      The Save Technology File form appears.
   h. Set the *Technology Library* cyclic field to the technology library you want to save.
   i. Click *OK*.

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 231 for information about assigning a different display packet to a layer.

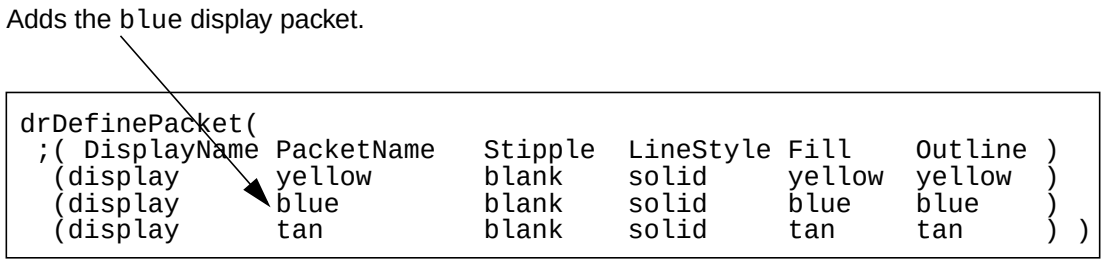
Creating a New Display Packet

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.

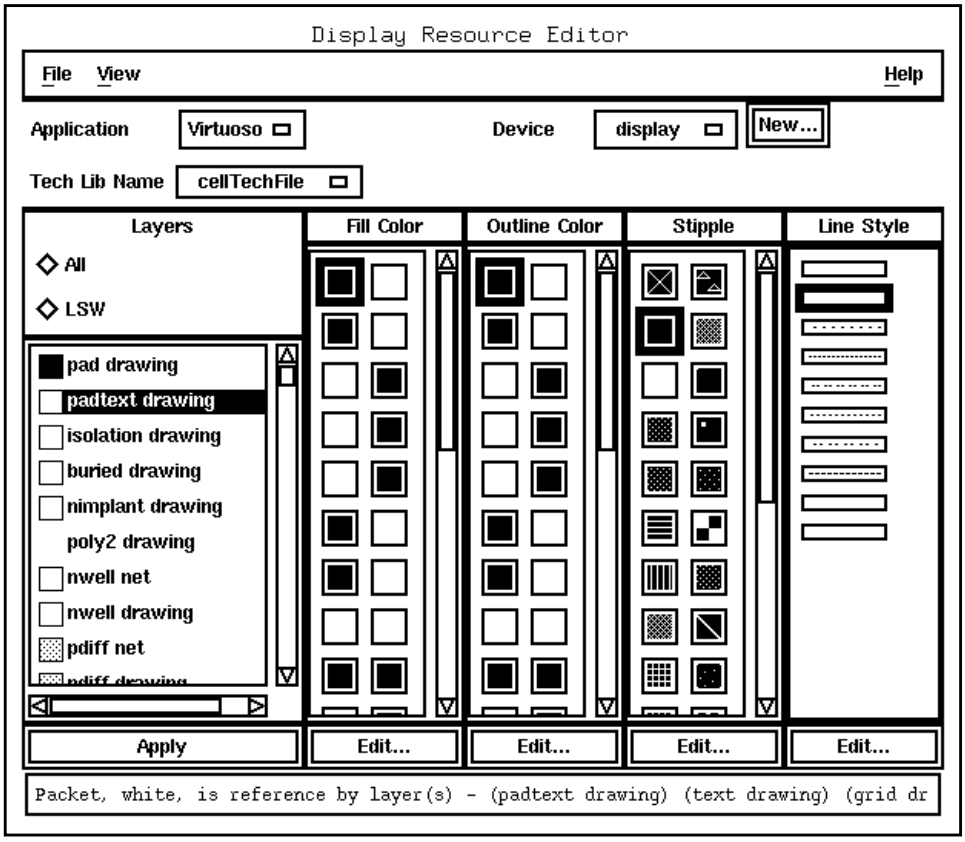
   The easiest way is to copy one of the lines and then change the display packet name and definition.

   In the following example, the `blue` display packet was added.



3. Save the `display.drf` file and exit the text editor.
4. From the Display Resources Tool Box, choose *Edit*.

The Display Resource Editor form appears.



5. Set the *Application* cyclic field to *DRE*.
6. *Choose File – Load*.

   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).

11. Close the Display Resource Editor form by choosing *File – Exit*.

D

Technology File and Display Resource File Examples

This appendix contains

- Technology File Example
- Display Resource File Example

Technology File Example

The following is a example of an ASCII technology file.

```
; Generated on Nov 14 07:50:20 1996
;     with layoutPlus version 4.4.1 Wed Nov 13 22:29:27 PST 1996 (cds3003)

;*******************************
; 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(
```

```
 ;layers are ordered from lowest to highest priority
 ;( LayerName                 Purpose    )
 ;( ---------                 -------    )
  ( nimplant                  drawing    )
  ( nwell                     net        )
  ( nwell                     drawing    )
 ) ;techLayerPurposePriorities

 techDisplays(
 ;( LayerName   Purpose  Packet  Vis  Sel  Con2ChgLy  DrgEnbl  Valid )
 ;( ---------   -------  ------  ---  ---  ---------  -------  ----- )
  ( nimplant    drawing  cyan    t    t    t          t        t     )
  ( nwell       net      yellow  t    t    nil        t        nil   )
  ( nwell       drawing  yellow  t    t    t          t        t     )
 ) ;techDisplays

techLayerProperties(
;( PropNameLayer1          [Layer2]PropValue )
 ( defaultWidthndiff    1.000000 )
 ( defaultWidth(pdiff drawing)1.000000 )
)

) ;layerDefinitions

;*******************************
; LAYER RULES
;*******************************
layerRules(

 streamLayers(
 ;( layer       streamNumber    dataType        translate )
 ;( -----       ------------    --------        --------- )
  ( ("ptap" "drawing") 0        0               nil       )
  ( ("ptap" "net")     0        0               nil       )
  ( ndiff              0        0               nil       )
 ) ;streamLayers

 viaLayers(
 ;( layer1      viaLayer        layer2     )
 ;( ------      --------        ------     )
  ( poly1       cont            metal1     )
```

```
  ( 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 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 )
 ) ;orderedSpacingRules

 spacingRules(
 ;( rule                layer1    layer2    value    )
 ;( ----                ------    ------    -----    )
  ( minSpacing          "ndiff"     1.0  )
  ( minSpacing          "pdiff"     1.0  )
 ) ;spacingRules
```

```
 mfgGridResolution(
      ( 0.100000 )
 ) ;mfgGridResolution
) ;physicalRules

;*******************************
; COMPACTOR RULES
;*******************************
compactorRules(
compactorLayers(
 ;( layer                    usage      )
 ;( -----                    -----      )
  ( ndiff                    "diffusion" )
  ( pdiff                    "diffusion" )
 ) ;compactorLayers
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


) ;compactorRules

;*******************************
; LE RULES
;*******************************
leRules(

 leLswLayers(
 ;( layer               purpose         )
 ;( -----               -------         )
  ( metal1Res           drawing         )
  ( text                drawing         )
 ) ;leLswLayers
) ;leRules
;*******************************
; P&R RULES
;*******************************
prRules(
```

```
 prRoutingLayers(
 ;( layer                       preferredDirection  )
 ;( -----                       -----------------   )
  ( poly1                       "halfRoute" )
  ( metal1                      "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       chocolate      128    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  )
)
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)
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
```

```
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
                              (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
                              (0 0 0 0 0 0 0 0 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)
                             ) )
( 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)
                              (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)
                              (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 0 1 0 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 0 1 0 0 0 1 0 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 0 1 0 0 0 1 0 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 0 1 0 0 0 1 0 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 1 0 0) )
 ( display       dots           1      (1 0 0) )
 ( display       dashDot        1      (1 1 1 0 0 1 0 0) )
 ( display       shortDash      1      (1 1 0 0) )
 ( display       doubleDash     1      (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" "metal1"     "bluevZigZag_S")
drDefinePacketAlias( "psb" "metal2"     "bluevZigZag_S")
drDefinePacketAlias( "psb" "net"        "bluevZigZag_S")
drDefinePacketAlias( "psb" "net2"       "bluevZigZag_S")
```