Data Formats

Format Definitions

L-Edit utilizes twelve different file formats. Following is a list of the formats and a brief description of each. For more information on a format, click on the link for each file type.

- **CAP** – Nodal properties files
- **CIF** – Caltech Intermediate Form files
- **DRC** – Design rule errors text file
- **EXT** – Extract definition files
- **GDS II** – Stream files
- **INI** – Application configuration files
- **RUL** – Design rule text files
- **SPC** – Extract netlist files
- **TDB** – Tanner database files
- **TPR** – Tanner Place and Route files
- **TTX** – Tanner text files
- **XST** – Cross-Section process definition files

CAP Files

Syntax

Note: The following text file is broken into two sections here to ensure readability on the page. In an actual CAP file, all of the information pertaining to an individual node is on a single line.

```
$ -----------------------------------------------------------------------------------
$ Nodal Properties File:  C:\documentation\bargraph.cap
$ SPR Date and Time   :  04/01/1998 - 07:01
$
$ 1 LU = 1/1 Lambda
$ 1 LU = 1/1 Micron(s)
$
$ Syntax:
$ Node      Capacitance    NoOf          M1Area        M1Length      M1Area
$                          Terminals     NoOvlap       NoOvlap       M2Ovlap
$           (1/100 pF)                   (LU^2)        (LU)          (LU^2)
$ -----------------------------------------------------------------------------------

N66         100            9             27039.0000    9013.000      1143.0000
N2          27             2             6705.0000     2235.000      396.0000
N4          27             2             4470.0000     1490.000      180.0000
N6          16             2             3951.0000     1317.000      126.0000
N8          30             2             6117.0000     2039.000      243.0000
N10         18             2             4908.0000     1636.000      180.0000
```

```
:

[continued]

           M1Length       M2Area         M2Length       M2Area         M2Length
           M2Ovlap        NoOvlap        NoOvlap        M1Ovlap        M1Ovlap
           (LU)           (LU^2)         (LU)           (LU^2)         (LU)
-----------------------------------------------------------------------------------

[N66]      381.000        5932.4000      1977.467       3529.0000      1176.333
[N2]       132.000        2423.0920      807.697        641.0000       213.667
[N4]       60.000         3567.7240      1189.241       1916.0000      638.667
[N6]       42.000         1622.5400      540.847        619.0000       206.333
[N8]       81.000         3509.3560      1169.785       1724.0000      574.667
[N10]      60.000         1673.1720      557.724        414.0000       138.000

:
```

Interpretation

Each line in the file is in the format:

```
node   nodal_capacitance   #_of_terminals
        M1_area_no_overlap   M1_length_no_overlap
        M1_area_M2_overlap   M1_length_M2_overlap
        M2_area_no_overlap   M2_length_no_overlap
        M2_area_M1_overlap   M2_length_M1_overlap
```

where ***node*** is the name of the node, ***nodal_capacitance*** is an integer denoting capacitance on the node in hundredths of a picofarad, and ***#_of_terminals*** is the number of pins attached to this node. ***M1_area_no_overlap***,

M1_length_no_overlap, ***M1_area_M2_overlap***, and ***M1_length_M2_overlap*** denote the length and area of the route taken by this node on layer *Metal 1* without and with overlaps to any other route in layer *Metal2*. ***M2_area_no_overlap***, ***M2_length_no_overlap***, ***M2_area_M1_overlap***, and ***M2_length_M1_overlap*** denote the length and area of the route taken by this node on layer *Metal 2* without and with overlaps to any other route in layer *Metal 1*.

For a detailed description of how nodal capacitances are calculated, see Output Options.

CIF Files

Caltech Intermediate Form (CIF) is a standard, machine-readable format for representing IC layout. CIF files are typically saved with the **.cif** extension.

Importing and Exporting

CIF files are loaded with **File > Import Mask Data** and saved with **File > Export Mask Data**, and by selecting CIF in the **Import/Export file type** drop-down menu. Unlike previous versions of TDB files, which are saved with the **.tdo** extension, backup files of previous CIF files are not created. Instead, when you try to write to an existing CIF file, L-Edit presents a warning about overwriting the file.

Geometry on hidden layers, or layers without legal CIF names, cannot be written out in CIF format. If this is attempted, then a warning appears.

Interpretation

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

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

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

Symbols

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

```
DS nnn a b;
```

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

The optional scaling factors **a** and **b** are applied to the integer coordinates and distances within a symbol by multiplying each value by **a** and then dividing the result by **b**. Scaling helps to shorten the length of CIF files by eliminating trailing zeros. By default, coordinates and distances in CIF are specified in units of 0.01 micron; **a** = 100 and **b** = 1 would allow values to be specified in microns instead. The coordinates (10,6) with **a** = 100 and **b** = 1, for example, are equivalent to

(1000,600) with **a** = 1 and **b** = 1. If **a** and **b** are not specified, then they are both assumed to be 1, and all integers are mapped to the 0.01 micron standard.

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

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

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

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

Calls (Instances)

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

```
C integer transformation;
```

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

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

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

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

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

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

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

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

calls symbol 44 and rotates its *x*-axis by 90°.

Geometric Primitives

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

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

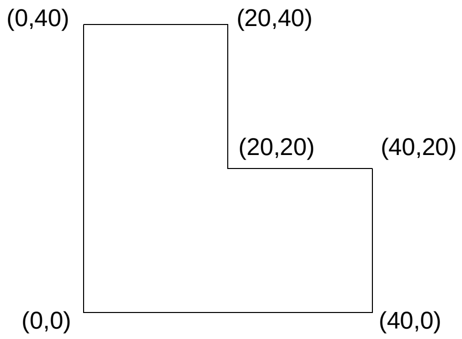
```
B 25 60 80,40 -20,20;      (box command);
```

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

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

```
P 0,0 0,40 20,40 20,20 40,20 40,0;
```

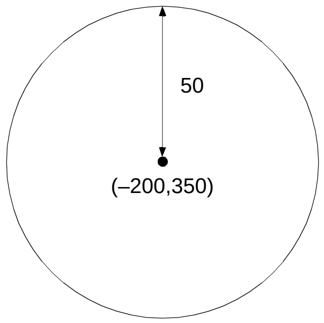
describes an L-shaped polygon with vertices at (0,0), (0,40), (20,40), (20,20), (40,20), and (40,0).



The **R** (roundflash) command defines a roundflash (circle) of fixed diameter and position. For example:

```
R 100 -200,350;     (roundflash command);
```

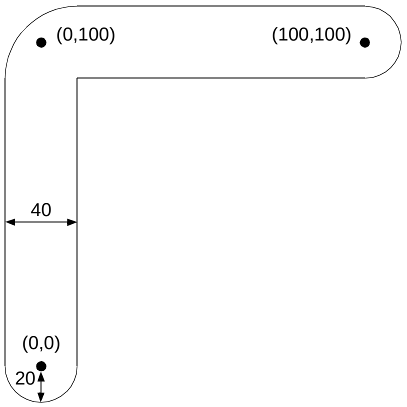
describes a circle of diameter 100 with center at (–200,350).



The **W** (wire) command defines a wire with fixed width along a specified path. A wire can be described as a long run of uniform width; ideally, the locus of points within one-half width of the given centerline or path and one-half width of the endpoints (semicircular caps). For example:

```
W 40 0,0 0,100 100,100;    (wire command);
```

describes a wire of width 40 with centerline vertices at (0,0), (0,100), and (100,100).



Layers

All primitive geometry elements must be associated with a particular fabrication mask or technology layer. Layers are specified with the **L** (layer) command. Primitives created after an **L** command belong to that layer until the layer is reset by the next **L** command. The form of the **L** command is:

```
L shortname; (layer command);
```

where **shortname** is the 1–4 character layer name. Layer names must be unique and correspond to fabrication masks being constructed. You should therefore take care that the layer names you use accord with the conventions established by your fabricator. **Setup Layers – General** correlates CIF layer names and technology layers; the CIF names are used instead of the L-Edit layer names during the conversion of the design file into CIF. Layer names that do not conform to legal CIF syntax must be modified before saving. Layer name specifications are preserved across symbol calls.

Layer names in the setup file must agree with the layer names of CIF files read in; otherwise, the geometry information on the non-matching layers in the CIF file will be transferred to the Icon layer. Your fabricator may apply additional restrictions and extensions to the CIF standard.

Restrictions

One piece of information which must be supplied to your fabricator is the name of the cell which represents the top level of your design. The fabricator will typically choose the top-level cell in your design, if it is the only such cell. However, if you do not specify this information and your fabricator has a choice about which cell to fabricate, the wrong one might be chosen.

L-Edit does not accept geometry other than CIF symbols. A CIF call (instance) to the top level of a design is achieved with **Cell > Fabricate**. **Fabricate** causes a CIF **C** command (or call to the selected cell) to be created at the top level, effectively identifying that cell as the cell to be fabricated. L-Edit only allows a single call outside of a symbol definition. If any rotations or transformations are embedded in this outside call, L-Edit suppresses them when the file is read.

Warning: Once a fabricate cell has been chosen, it will remain the fabricate cell until a new one is chosen, even if it ceases to be the top-level cell in your design. Be sure to check the fabricate cell before writing a CIF file!

L-Edit accepts forward references (symbol calls before the symbol definitions they reference). L-Edit also removes forward references during conversion of the design into CIF.

L-Edit does not support the CIF **DD** (delete symbol definition) command.

Extensions

L-Edit supports two user extensions to the basic CIF syntax. The first extension is a cell name extension of the form:

```
9 cellname;
```

where **cellname** is the name of the currently open CIF symbol. This command can only appear within the context of an open symbol (between a **DS**/**DF** command pair). The cell name may contain spaces and must be terminated with a semicolon. Duplicate, zero-length, and null cell names are not permitted.

If a CIF file does not define cell names for CIF symbols, then L-Edit automatically assigns as the cell name the expression:

```
(DS nnn)
```

where ***nnn*** is the CIF symbol number. This definition is suppressed when the CIF file is written out. You should therefore avoid naming cells with this syntax, or else the name will be suppressed during CIF file conversion. L-Edit reads out-of-order cell numbers, but always orders cells by number while writing out the design in CIF.

The second user extension is a port extension of the form:

```
94 portname x y layer;
```

where ***portname*** is the name of the port (label), ***x*** and ***y*** are the coordinates of the port, and **layer** is the name of the port's layer. This is a relatively standard port or label user extension to CIF. However, it is not as flexible as L-Edit's definition of a port. An L-Edit port can be a point, a line, or a box, and the text can be rotated in a variety of ways; this CIF user extension can only represent a single point, with no information on the position or rotation of the associated text. When L-Edit writes a port into a CIF file, it computes the centerpoint of the port and records this in the CIF file as the position of the port. You can preserve the box associated with the port in CIF as written by L-Edit by checking the **Write Port-boxes** option in the **File > Export Mask Data**, **Options** button dialog. This results in the use of nonstandard notation for ports, and other software tools may not be able to read this form of CIF. To read this CIF back into L-Edit, you must check the **Read rectangular polygons as boxes** option in the **File > Import Mask Data**, **Options** button dialog before reading the file.

Wires

CIF was developed at a time when masks were usually created by Gerber photoplotters. Such plotters could make wires by opening a circular aperture and moving it along a pathway. The resulting wire would therefore have rounded corners and ends. This fabrication method gave rise to the CIF specification for rounded wires. However, present-day mask making is almost entirely raster-based, and thus has a strong affinity toward orthogonal structures. So many fabricators assume CIF wires to have extended wire end styles with mitered corners. Thus to adhere to the fabricators' implementation of wires, all your CIF wires should be of extend end style and layout join style. Upon fabrication, many fabricators such as MOSIS and Orbit run both CIF and GDS II files through CATS (a high-end program used by many fabricators and mask houses to produce formats for specific mask-making equipment from GDS II and CIF layout files). CATS uses its own clipping algorithm for acute angle CIF wires and GDS II paths with a pathtype of 0 or 2. This algorithm corresponds exactly with the L-Edit wire layout join style, the default wire join style, which employs a miter length of one-half the width of the wire for wires with an acute join angle. You should check with your fabricator concerning the exact method of fabrication used for wires before using wires in your layout.

Scaling

Apart from the user-selectable scaling of L-Edit’s internal units, L-Edit incorporates an implicit scaling factor while writing CIF files. Due to the manner in which geometric objects are represented in CIF, it is necessary for L-Edit to apply an implicit multiplication factor of two to all geometry as it is written out to CIF. The reason for this scaling is that CIF represents boxes with integer length, width, and center coordinates. L-Edit, however, can create boxes with fractional center coordinates: a box of width and length 3 with lower left corner at (0,0) has its center at (1.5,1.5), for example. L-Edit circumvents this problem by multiplying all coordinates by two when writing a CIF file. The same box, after being written out to a CIF file, would have a length and width of 6 and be centered at (3,3). L-Edit incorporates this multiplication by 2 into the scaling factors recorded in the CIF file, so that when the file is read in by a CIF reader it is scaled correctly.

DRC Files

When the **Write errors to file** checkbox in the **Design Rule Check** dialog is checked, design rule errors are written to a text file with the extension **.drc**. These files can be opened with any text editor.

For more information on DRC files, see Error Files.

EXT Files

The extract definition file contains a list of comments, connection statements, and device statements, with the following restrictions:

- Layer names are case-sensitive, and must match the case of layer names defined in the TDB file. The rest of the definition file is case-insensitive; upper and lower cases can be used interchangeably.
- Layer names cannot contain commas or semicolons and they cannot be longer than 40 characters.
- Layer names cannot have leading or trailing spaces.
- Pin names cannot contain commas, semicolons, or spaces, and they cannot be named **MODEL**.
- Model names cannot contain commas, semicolons, spaces, or closing parentheses.
- For compatibility with existing extract definition files, the *WIDTH* parameter is ignored for all devices except a GAASFET/MESFET.
- IGNORE_SHORTS indicates that if the device has all of its pins connected to the same node then it will be considered shorted and the device will be written to the extract netlist file as a comment.

Comment Statements

A comment statement begins with a pound sign (#) and continues to the end of the line:

```
# This is an extract definition file comment.
```

Connection Statements

A connection statement defines a connection between two different process layers. A connection always involves three layers: the two layers being connected and the “via” or “contact” layer which connects them. Connection statements have the following format:

CONNECT (*Layer1*, *Layer2*, *ThroughLayer*)

where **Layer1** and **Layer2** are the names of the layers being connected, and **ThroughLayer** is the name of the connecting layer. For example:

```
# Connect Poly to Metal1
CONNECT (Poly, Metal1, PolyContact)
```

Device Statements – General Format

A device statement defines a device. Both passive (capacitors, resistors, and inductors) and active (BJTs, diodes, GaAsFETs, JFETs, MOSFETs, and subcircuits) devices are specified with the same general format.

All device statements require that a *recognition layer* — one of the layers involved in the construction of the device — be identified. You may specify multiple devices with the same recognition layer (as long as they have different pins). This is particularly useful in extracting multi-source/drain transistors. The recognition layer is defined as follows:

```
RLAYER = rLayer ;
```

where **RLAYER =** is required, and **rLayer** is the name of the recognition layer.

Following the recognition layer is a list of pins on the device. The order of this list determines the order the pins will be in the extracted netlist. The extractor does not require any particular order, but LVS requires that both source netlists contain pins in the same order, and SPICE simulators also have strict rules about the order in which pins appear. We recommend following the standard SPICE orders:

- BJT devices: collector —þbase —þemitter —þsubstrate
- All other active devices: drain — gate —þsource —þbulk

If the pin names used are **Collector**, **Base**, **Emitter**, and **Substrate** (BJT devices), or **Drain**, **Gate**, **Source**, and **Bulk** (all other active devices), then they are sorted automatically.

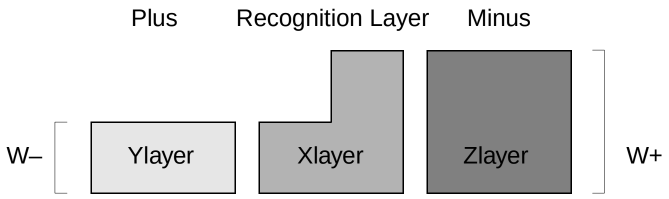
Pins are specified as follows:

```
pinName = pinLayer ;
```

where **pinName** is the name of the pin and **pinLayer** is the name of the associated layer.

Extract uses the formula $R = \rho \times (l / w)$ for calculating the value of the extracted resistance, where ρ is the sheet resistance in units of ohms/square, *l* is the length, and *w* is the width. The value of ρ is taken from the number specified with **Setup > Layers** for the recognition layer of the resistor. The values of *l* and *w* are determined from the layout.

The extractor computes the area of the recognition layer and divides it by the effective width to obtain *l*. The effective width is the average of *W*+ and *W*–.



Following the list of pins is a model definition. This definition is not required (**MODEL =;** is acceptable). The model name, if present, will be written into the extracted netlist. For SPICE, model names are not generally required for capacitors, resistors, inductors, or diodes, but are required for all other devices. Model statements have the form:

```
MODEL = [ModelName] ;
```

where **MODEL =** is required and **ModelName** is the optional model name. The empty statement **MODEL =;** is still required if no model name is specified.

Device Statements – Specific Formats

In the following format specifications:

- Unitalicized words and characters (except brackets **[ ]**) are to be entered as shown.
- Variables containing the string **Layer** represent layer names.
- **ModelName** represents the SPICE model name for the device.

Capacitor

```
DEVICE=CAP (
      RLAYER = rLayer ;
      Plus = Layer1 ;
```

```
       Minus = Layer2 ;
       MODEL = ModelName ;
) [IGNORE_SHORTS]
```

A capacitor has the following format in the SPICE output statement:

```
       Cxxx n1 n2 ModelName [C=]cValue
```

The following rules apply to capacitors:

- The capacitance will be based on the area of the recognition layer (*rLayer*).
- Capacitance is calculated as follows:

$C_{total} = C_{area} + C_{fringe}$

C_{area} = (Area of Layer) * (Layer Area Capacitance)

C_{fringe} = (Layer Perimeter) * (Layer Fringe Capacitance)

- The fringe capacitance (fF/micron) and area capacitance (aF/sq. micron) are specified in the **Setup > Layers** dialog for each specific layer.

Resistor

```
DEVICE=RES (
       RLAYER = rLayer ;
       Plus = Layer1 ;
       Minus = Layer2 ;
       MODEL = ModelName ;
) [IGNORE_SHORTS]
```

A resistor has the following format in the SPICE output statement:

```
      Rxxx n1 n2 ModelName [R=]rValue
```

The following rules apply to resistors:

- The resistance is calculated based on the area of the recognition layer (*rLayer*) and the widths of the edges of the **Plus** pin and **Minus** pin that touch the recognition layer (*rLayer*).
- Resistance is calculated as follows:

  ```
  R = ρ * (length/width)
  ```

- The sheet resistance ρ (ohms/square) is specified in the **Setup > Layers** dialog for the recognition layer.

Inductor

```
DEVICE=IND (
      RLAYER = rLayer ;
      Plus = Layer1 ;
      Minus = Layer2 ;
      MODEL = ModelName ;
) [IGNORE_SHORTS]
```

An inductor has the following format in the SPICE output statement:

```
        Lxxx n1 n2 ModelName [L=]
```

No inductance value is calculated by the Extract module.

BJT

```
DEVICE=BJT (
        RLAYER = rLayer [,AREA] ;
        Collector = cLayer ;
        Base = bLayer ;
        Emitter = eLayer ;
        Substrate = sLayer ;
        MODEL = ModelName ;
        NominalArea = areaVal ;
) [IGNORE_SHORTS]
```

A BJT device has the following format in the SPICE output statement:

```
        Qxxx nc nb ne ModelName [AREA=pinArea/areaVal]
```

The following rules apply to BJT devices:

- Nominal area can be expressed either in decimal or scientific notation, and has units of m^2 but no unit tag will appear after the value.

- If no AREA keyword is present, the area will not be written to the SPICE statement.

Diode

```
DEVICE=DIODE (
        RLAYER = rLayer [, AREA] ;
        Plus = Layer1 ;
        Minus = Layer2 ;
        MODEL = ModelName ;
        NominalArea = areaVal ;
) [IGNORE_SHORTS]
```

A diode has the following format in the SPICE output statement:

```
        Dxxx n1 n2 ModelName [AREA=pinArea/areaVal]
```

The following rules apply to diodes:

- Nominal area can be expressed either in decimal or scientific notation, and has units of m^2 but no unit tag will appear after the value.
- If no AREA keyword is present, the area will not be written to the SPICE statement.

GAASFET/MESFET 1

```
DEVICE=GAASFET (
       RLAYER = rLayer [, AREA] ;
       Drain = dLayer ;
       Gate = gLayer ;
       Source = sLayer ;
       Bulk = bLayer ;
       MODEL = ModelName ;
       NominalArea = areaVal ;
) [IGNORE_SHORTS]
```

A GAASFET/MESFET device has the following format in the SPICE output statement:

```
       Zxxx nc nb ne ModelName [AREA=pinArea/areaVal]
```

The following rules apply to GAASFET/MESFET devices:

- Nominal area can be expressed either in decimal or scientific notation, and has units of m^2 but no unit tag will appear after the value.
- If no AREA keyword is present, the area will not be written to the SPICE statement.
- GAASFET/MESFET definition is distinguished only by the presence of the AREA or WIDTH keyword. L-Edit determines the appropriate output based on the keyword.

GAASFET/MESFET 2

```
DEVICE=GAASFET (
       RLAYER = rLayer ;
       Drain = dLayer [, WIDTH] ;
       Gate = gLayer ;
       Source = sLayer [, WIDTH] ;
       Bulk = bLayer ;
       MODEL = ModelName ;
) [IGNORE_SHORTS]
```

A GAASFET/MESFET device has the following format in the SPICE output statement:

```
       Zxxx nd ng ns ModelName L=length W=width
```

The following rules apply to GAASFET/MESFET devices:

- The length is the length of gate, and the width is the width of the indicated layer in contact with the gate. The length and width have units of meter.
- The optional WIDTH parameter for a GAASFET/MESFET may be specified on only the drain or source pin but not both, and is used to indicate the layer for which width will be calculated.
- GAASFET/MESFET definition is distinguished only by the presence of the AREA or WIDTH keyword. L-Edit determines the appropriate output based on the keyword.

- If no WIDTH keyword is present, the width and length will not be written to the SPICE statement.

JFET

```
DEVICE=JFET (
      RLAYER = rLayer [, AREA] ;
      Drain = dLayer ;
      Gate = gLayer ;
      Source = sLayer ;
      Bulk = bLayer ;
      MODEL = ModelName ;
      NominalArea = areaVal ;
) [IGNORE_SHORTS]
```

A JFET device has the following format in the SPICE output statement:

```
      Jxxx nd ng ns ModelName [AREA=pinArea/areaVal]
```

The following rules apply to JFET devices:

- Nominal area can be expressed either in decimal or scientific notation, and has units of m^2 but no unit tag will appear after the value.
- If no AREA keyword is present, the area will not be written to the SPICE statement.

MOSFET

```
DEVICE=MOSFET (
       RLAYER = rLayer ;
       Drain = dLayer [, AREA] [, PERIMETER[/GATE=#]] ;
       Gate = gLayer ;
       Source = sLayer [, AREA] [, PERIMETER[/GATE=#]] ;
       Bulk = bLayer ;
       MODEL = ModelName ;
) [IGNORE_SHORTS]
```

A MOSFET device has the following format in the SPICE output statement:

```
       Mxxx nd ng ns nb ModelName L=lengthValue W=widthValue
       [AD=areaValue] [PD=perimeterValue] [AS=areaValue]
       [PS=perimeterValue]
```

The following rules apply to MOSFET devices:

- The length is the length of gate, and the width is the average of the width of the source and drain in contact with the gate. The length and width have units of meter.
- The optional AREA parameter for a MOSFET may be specified on the drain or source pin or both, and is used to indicate whether the area for that layer will be calculated and written for the AD (Drain Area) and AS (Source Area) output values.

- The optional PERIMETER parameter for a MOSFET may be specified on the drain or source pin or both, and is used to indicate whether the perimeter for that layer will be calculated and written for the PD (Drain Perimeter) and PS (Source Perimeter) output values.
- The optional /GATE=# parameter that is used with the PERIMETER parameter for a MOSFET may be specified on the drain or source pin or both, but only where the PERIMETER parameter has already been designated. The number is a floating point value between 0.0 and 1.0, indicating the fraction of gate width to include in the perimeter. If the /GATE=# parameter is missing, the perimeter will include the gate width.

Subcircuit

Subcircuits can be defined explicitly for the extractor. This method of describing subcircuits is different from automatic subcircuit instance recognition (see Subcircuit Recognition).

```
DEVICE=SUBCKT (

       RLAYER = rLayer [, AREA] ;
       pinName = pinLayer [, AREA] ;
       pinName = pinLayer [, AREA] ;
       . . .
       MODEL = ModelName ;
       NominalArea = areaVal ;
) [IGNORE_SHORTS]
```

A subcircuit has the following format in the SPICE output statement:

```
Xzzz n1 n2 n3 ... cName [AREA=rLayerArea/areaVal]
[AREA_pinName=pin1Area/areaVal]
[AREA_pinName=pin2Area/areaVal] ...
```

The following rules apply to subcircuits:

- The optional AREA parameter for a subcircuit may be specified on one or more layers and an area will be calculated for the indicated layer.
- Nominal area can be expressed either in decimal or scientific notation, and has units of m^2 but no unit tag will appear after the value.
- If no AREA keyword is present, the area will not be written to the SPICE statement.

GDS II Files

GDS II (stream) is a standard, machine-readable format for representing IC layout. GDS II files are typically saved with the **.gds** extension.

Importing and Exporting

GDS II files are loaded with **File > Import Mask Data** and saved with **File > Export Mask Data**, and by selecting GDSII in the **Import/Export file type** drop-down menu. Unlike previous versions of TDB files, which are saved with the **.tdo** extension, backup files of previous GDS II files are not created. Instead, when you try to write to an existing GDS II file L-Edit presents a warning about overwriting the file.

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

Geometry on hidden layers cannot be written out in GDS II format. If this is attempted, then a warning appears.

Interpretation

GDS II stream format is a binary file format for interchanging mask geometry information between different IC CAD systems. The L-Edit implementation of GDS II file reading and writing conforms to the Calma Stream Format, GDS II release 3.0, with some limitations.

A GDS II file may contain a single design or a library of designs. GDS II assumes right-handed geometry, with the *x*-axis increasing to the right and the *y*-axis increasing upward. The basic unit is set to the GDS II default (user unit = 1 micron and 1000 database units per user unit).

Most L-Edit elements have a one-to-one correspondence with elements of GDS II stream files. GDS II last access time information is not supported by L-Edit. L-Edit circles are approximated by GDS II polygons. L-Edit cell names may be modified going to GDS II.

The table below shows the correspondence between L-Edit elements and their GDS II names. GDS II data types for L-Edit boxes, wires, and polygons can be viewed and edited in the **Edit Object(s)** dialog with **Edit > Edit Object(s)**.

L-Edit	***GDS II***
File	Stream file
Cell Definition	Structure

L-Edit	*GDS II*
Box	Boundary *
Box **	Box
Polygon	Boundary
Wire	Path
Circle	Boundary ***
Instance	SRef
Array	ARef
Port	Text
Data type	Data type

* L-Edit boxes are written to GDS II files as 4-sided boundaries (polygons). When reading boundaries from a GDS II file, L-Edit checks each one to see if it is a 4-sided orthogonal polygon, and if so, represents it as an L-Edit box.

** GDS II boxes are not intended to be mask geometry and are generally discarded by mask-making software. If L-Edit encounters GDS II boxes while reading a GDS II file, a dialog is presented with two options: discard all GDS II boxes or convert them to L-Edit boxes (mask geometry).

*** L-Edit circles are written by default as 64-sided polygons.

GDS II allows only the following restricted set of characters in cell names. “a” … “z”, “A” … “Z”, “0” … “9”, underscore “_”, question mark “?”, and dollar sign “$”. L-Edit cell names may include a much richer set of characters, some of which would be illegal in GDS II. Therefore, L-Edit checks each cell name before writing it out to a GDS II file. If any spaces “ ” are found, then L-Edit replaces them with underscores “_” in the GDS II file. If any other illegal characters are found, then L-Edit requests that you change the name.

Some GDS II systems do not recognize lower case letters in cell names. For interfacing with these systems, L-Edit provides the capability to write all cell names to a GDS II file in upper case. This option is enabled by a check box in the **File > Export Mask Data**, **Options** button dialog.

GDS II does not contain a specification for circles. Therefore, L-Edit approximates circles using 64-sided polygons. Thus, circles are not preserved through writing a GDS II file and reading it back in.

L-Edit supports all-angle rotations of instances (in integer degrees; fractional angles are rounded without warning) and 90° rotations of text.

Due to the treatment of 4-sided polygons upon being read in, L-Edit polygons that happen to be orthogonally oriented rectangles will not be preserved through writing to a GDS II file and reading back in. These special polygons will come back into L-Edit as boxes. Of course, from the standpoint of the mask that gets fabricated, there is no difference between a box and its equivalent polygon.

Many different versions of GDS II readers and writers exist. Some newer versions produce elements which are not compatible with older versions of GDS II. The elements in L-Edit are confined to elements which are common to all.

Wires

The GDS II layout format allows for three different types of wires (paths): paths with butt ends and square corners, paths with extended ends and square corners, and paths with round ends and round corners. The three GDS II pathtypes correspond to three of the twelve possible L-Edit wires. When reading GDS II paths, L-Edit sets end styles and join styles appropriate for the three GDS II pathtypes. When creating GDS II output, L-Edit chooses the GDS II pathtype according to the following table. (Some of the wire-to-path conversions are accompanied on output by a warning message; these are indicated in the table with an asterisk *.)

End Style	*Join Style*	*GDS II Pathtype*
Butt	Layout	0
Butt	Miter	0 *
Butt	Round	0 *
Butt	Bevel	0 *
Round	Layout	1 *

End Style	*Join Style*	*GDS II Pathtype*
Round	Miter	1 *
Round	Round	1
Round	Bevel	1 *
Extend	Layout	2
Extend	Miter	2 *
Extend	Round	2 *
Extend	Bevel	2 *

Upon fabrication, many fabricators such as MOSIS and Orbit run GDS II files through CATS (a high-end program used by many fabricators and mask houses to produce formats for specific mask-making equipment from GDS II layout files). CATS uses its own clipping algorithm for acute angle GDS II paths with a pathtype of 0 or 2. This algorithm corresponds exactly to the L-Edit layout wire join style, the default wire join style. Layout join style employs a fixed miter length of one-half the width of the wire for wires with an acute join angle.

When you are about to use wires for the first time or you are setting up the technology files for others who may use wires, take a moment to set up the wire defaults for each layer according to whether your likely output format will be GDS II. For GDS II, use one of the three legitimate combinations of end style and join style. It is also strongly recommended that you contact your fabricator before you define the wire styles for your design and understand how they will interpret GDS II wires.

INI Files

Application configuration (INI) files save application settings. These files are specified with **Setup > Application**. All parameters on the **Setup Application — General** tab and changes made to keyboard mapping are saved to INI files.

Following is the list of parameters saved in INI files.

Parameter	*More information*
Keyboard remapping	**Setup Application – Keyboard**
Editing options	**Setup Application – General**
TDB setup path	**Setup Application – General**
Toolbar settings	**Setup Application – General**
Recently used file list size	**Setup Application – General**
UPI macro files loaded at startup	Tools> Macro

INI files use the Windows INI file format and can be edited with any text editor.

Workgroup and User Files

Information from an INI file is loaded into L-Edit as either a **Workgroup** or a **User** file. **Workgroup** files are intended to be shared by multiple users; for example, they may contain key remapping sequences that will be used by many users. **User** files are intended to contain preferences specific to a particular user. It is most likely that a **Workgroup** file would reside on a network, while a **User** file would reside on an individual user's machine.

Changes in the **Setup Application** dialog can only be saved to **User** configuration files. Therefore, an INI file loaded as a **Workgroup** file is protected from accidentally being changed.

RUL Files

When design rules are written to a file with the **Write to file** button in the **Setup Design Rules** dialog, the default extension on the file is **.rul**. RUL files are text files and can be opened with any text editor.

RUL files cannot be read by L-Edit's layout editor. To import a set of design rules into an existing L-Edit design file, use **File > Replace Setup** and the choose the appropriate TDB or TTX file.

For more information on RUL files, see Rule Lists.

SPC Files

SPC files are standard Berkeley 2G6 SPICE netlists. They can be used with Tanner EDA’s T-Spice circuit simulator, or with any other tools that read SPICE netlists.

Device Statements

Passive Devices

Passive element (capacitor, resistor, or inductor) statements have the following form:

```
Cxxx n1 n2 [ModelName] [C=]cValue
Rxxx n1 n2 [ModelName] [R=]rValue
Lxxx n1 n2 [ModelName] [L=]
```

xxx	Unique element name
n1 n2	Node names
ModelName	Device model name. Model names must be defined with **.model** commands.
cValue	Capacitance

rValue	Resistance

Extract does not write the inductance value.

For example:

```
        C1 N1997 SET1 C=120pF
```

This defines a 120 pF capacitor **C1** with one pin connected to node **N1997** and the other connected to node **SET1**.

Active Devices

Active or semiconductor device (diode, BJT, GAASFET/MESFET, JFET, or MOSFET) statements have the following form:

```
Dxxx n1 n2 ModelName [AREA=pinArea/areaVal]
Qxxx nc nb ne [ns] ModelName [AREA=pinArea/areaVal]
Zxxx nc nb ne ModelName [AREA=pinArea/areaVal]
Zxxx nd ng ns ModelName L=length W=width
Jxxx nd ng ns ModelName [AREA=pinArea/areaVal]
Mxxx nd ng ns [nb] ModelName L=lengthValue W=widthValue
        [AD=areaValue] [PD=perimeterValue] [AS=areaValue]
```

```
[PS=perimeterValue]
```

xxx	Unique element name
n1 n2	Diode node names
nc nb ne ns	Collector, base, emitter, and substrate node names (BJT devices)
nd ng ns nb	Drain, gate, source, and bulk node names (GAASFET/MESFETs, JFETs, and MOSFETs)
ModelName	Device model name. Model names must be defined with **.model** commands.
aValue	Area parameter
length	Length parameter
width	Width parameter

Parameters may appear in any order.

For example:

```
M12 17 19 21 21 PMOS L=2U W=28U
```

defines a PMOS transistor **M12**. The drain node is **17**, the gate node is **19**, and the source node and bulk nodes are the same, **21**. The transistor is 2 microns long and 28 microns wide.

Subcircuit Instances

A subcircuit is a list of devices and nodes which can be instanced repeatedly. Subcircuit instance statements have the following form:

```
Xzzz n1 n2 n3 ... cName [AREA=rLayerArea/areaVal]
[AREA_pinName=pin1Area/areaVal]
[AREA_pinName=pin2Area/areaVal] ...
```

zzz	Unique element name
n1 n2 n3 …	Node names. There must be as many node names listed as there are in the subcircuit definition.
cName	Subcircuit name
param1=value1 param2=value2 …	Parameters specified by the subcircuit definition. If a particular parameter is not specified on the device statement, then its default value is assumed from the subcircuit definition.

For example:

```
        X123 N125 N253 N74 myCircuit AREA=100 AREA_Pin1=15
```

This defines an instance **X123** of a subcircuit called **myCircuit**. It has three pins, connected to nodes **N125**, **N253**, and **N74**.

Device Commands

Subcircuits

The subcircuit definition command has the following form:

```
.SUBCKT cName pin1 [pin2 ...] [param1=value1]
       [param2=value2 ...]
       <subcircuit statements>
.ENDS [cName]
```

cName	Subcircuit name
pin1 pin2 …	Pin (input/output) names
param1=value1 param2=value2 …	Parameters

In between the first (**.subckt**) and last (**.ends**) lines are any number of SPICE device statements defining a functional unit. The only statements not allowed within a subcircuit definition are subcircuit and model commands. If the body of the subcircuit definition is empty, then the subcircuit must be defined in an element definition file to be used with LVS.

The extractor itself does not insert the subcircuit definition body between the **.subckt** and **.ends** lines. The SUBCKT mechanism has been adopted in Extract primarily to aid in doing LVS verification of non-standard (i.e., non-SPICE) elements such as CCDs. If subcircuit will be simulated, do not use the **.subckt** statements; if they are used, the appropriate body must be supplied to handle the subcircuit during simulation. Refer to the LVS manual for more information on how to utilize subcircuits during LVS verification.

Models

A model command defines a model name to be used in device statements. The model command can appear anywhere in the SPICE file, even after the model being defined is used in an element statement. The format of a model command is as follows:

```
.MODEL Modelname ...
```

where **Modelname** is the name of the model which is specified in the extract definition file.

For example:

```
.MODEL MYDEVICE
```

could be elsewhere in the netlist as:

```
M123 42 51 7 7 MYDEVICE L=2U W=28U
```

which defines a transistor **M123** using model **MYDEVICE**. Its drain is connected to node **42**, its gate to node **51**, and its source and bulk to node **7**. It has a length of 2 microns and a width of 28 microns.

End

Anything after the following command in a SPICE file is ignored:

```
.END
```

Comments

SPICE comment lines begin with an asterisk (*****).

Non-Standard Devices

The SPICE format used by Extract only allows for the devices described above. Non-standard devices (such as multi source/drain transistors and CCDs) are written as empty subcircuit definitions with an instance statement for each device. For simulation purposes the subcircuit definitions can be manually edited. For LVS comparison, you can specify the subcircuits as special devices in the element file.

TDB Files

Tanner Database (TDB) is a proprietary, machine-readable format optimized for the Tanner Tools environment. TDB files are typically saved with the .tdb filename extension and are opened with **File > Open**. By default, the scrollable list displays TDB files.

In addition to the design itself, a TDB file contains setup information including: layer rendering information, CIF and GDS II setup information, design rules, and L-Edit configuration settings. The TDB format can be read, displayed, and modified by L-Edit on any platform, and is the preferred format for storing L-Edit design information. The setup information can also be stored in Tanner Text (TTX) format and edited directly in a text editor. The setup information can be read back into L-Edit in either the TDB or the TTX format with **File > Replace Setup**.

TDB files are saved with **File > Save** or **File > Save As**. When a TDB file is saved, L-Edit automatically backs up previously-saved versions of the filename by preserving them with a .tdo extension.

TPR Files

Only netlist files in Tanner Place and Route (TPR) format can be used by L-Edit⁄ SPR to generate chip layouts. TPR files are ASCII text files that are generated automatically by the schematic editor S-Edit or they can be created with any text editor.

Syntax

A portion of the TPR netlist file for the bargraph example is shown below.

Comment line

```
$ TPR written by the Tanner Research schematic editor, S-Edit
$ Version: 2.0 Beta 5     Jan 7, 1998  16:07:16
```

Pad cell definition

```
CP PadOut DataOut Pad;
```

Instance definition

```
UPadOut_1 N2 PAD_B1_L31;

:

CP PadInC DataIn DataInB DataInUnBuf Pad;
UPadInC_1 N68 IPAD_9/N2 IPAD_9/N1 PAD_L9_SCO;
```

In the two lines above, **DataIn**, **DataInB**, and **DataInUnBuf** are the names of ports in the pad cell **PadInC** (PortList). **N68**, **IPAD_9/N2**, and **IPAD_9/N1** are the names of nets attached to these ports (NetList). **PAD_L9_SCO** is the name given to the body region of the pad. "**L9**" identifies the position of the pad as the ninth pad from the top on the left side of the padframe.

```
:
```

Ground pad

```
CP PadGnd Pad;
UPadGnd_1 PAD_R8_GND;
```

Power pad

```
CP PadVdd Pad;
UPadVdd_1 PAD_L6_VDD;

:
```

Cell definition

```
C INV A Out;
```

Instance definition

```
UINV_3 BARGRAPH_1/BG64_2/N9 BARGRAPH_1/BG64_2/SFT3;

:

C Mux2 A B Out Sel;
```

```
UMux2_1 BARGRAPH_1/BG64_1/BG4_1/N118 BARGRAPH_1/BG64_1/BG4_1/N108 N62

+  BARGRAPH_1/BG64_1/S11;
```

In the three lines above, **A**, **B**, **Out**, and **Sel** are ports in the standard cell **Mux2** (PortList). **BARGRAPH_1/BG64_1/BG4_1/N118**, **BARGRAPH_1/BG64_1/BG4_1/N108**, **N62**, and **BARGRAPH_1/BG64_1/S11** are the names of nets attached to these ports (NetList). Note that these net names include the hierarchical structure of the schematic. This is the manner in which S-Edit creates a “flattened” TPR netlist.

A plus sign (**+**) indicates a continuation of the previous line.

Interpretation

Pad cells are defined in the format:

```
CP <padname> <pin1> <pin2> … Pad
U<gateUID> <net1> <net2> … Pad_<PadPosition>
```

Standard cells are defined in the format:

```
C <cellname> <pin1> <pin2> …
U<gateUID> <net1> <net2> …
```

A TPR file must conform to the following rules:

- All signals which are to be routed within the core or from the core to the padframe are required to be listed, with the exception of the Vdd and Gnd signal connections to pads.
- For each cell, the PortList and NetList must have the same number of elements.
- The name “PAD” in the PortList of a pad cell refers to the actual bonding region of the pad, and is not actually involved in the placement and routing process. Pad cells must have a signal marked “PAD.”
- The bonding region of a pad can contain the location of the pad on the padframe. For example, “B1” stands for the leftmost pad on the bottom side of the padframe. (L = Left, B = Bottom, R = Right, T = Top.)
- Power and ground pads do not have to be included in the netlist. If they are not included, L-Edit ⁄ SPR will place them automatically.
- The parts listed in the file must match the cells contained in the layout library. To match, the name of the part must be identical to the name of the library cell (except for case), and every signal listed in the part description must have at least one port of the same name somewhere in the library cell.

TTX Files

Tanner Text (TTX) files contain setup information saved with **File > Export Setup**. The setup information can be read back into L-Edit with **File > Replace Setup**.

Syntax

The TTX format is organized by categories. Default values are assumed when categories are not specified.

A formal description of TTX syntax follows the table of variables used in the description. In the following table more than one value is possible for each of the string variables. Possible values are separated by vertical bars (|). Numbers can be written in either hexadecimal or decimal format unless specified otherwise.

Variable	*Type*	*Value*
C	***Numerical***	Color index number from 0–15
H	***Numerical***	Hex number
I	***Numerical***	Long integer
L	***Numerical***	Number of locator units

Variable	*Type*	*Value*
N	***Numerical***	Number
P	***Numerical***	Number of pixels
R	***Numerical***	Real number
U	***Numerical***	Number of internal units
V	***Numerical***	Color value number from 0–255
boolean	***String***	**TRUE** | **FALSE**
cursor	***String***	**SNAPPING** | **SMOOTH**
end	***String***	**BUTT** | **ROUND** | **EXTEND**
join	***String***	**MITER** | **ROUND** | **BEVEL** | **LAYOUT**
layer	***String***	[any valid layer name]
mode	***String***	**set** | **clear**
name	***String***	[any valid name]
operation	***String***	**AND** | **OR**
option	***String***	**SELECT** | **NOT SELECT**

Variable	*Type*	*Value*
rule	***String***	**MIN_WIDTH** | **EXACT_WIDTH** | **OVERLAP** | **EXTENSION** | **NOT_EXISTS** | **SPACING** | **SURROUND**
style	***String***	**ARROWS_AT_BOTH_END** | **NO_ARROWS**
text	***String***	**NO_TEXT** | **CENTERED** | **AT_END_POINTS** | **AT_TICK_MARKS**
unit	***String***	**microns** | **millimeters** | **centimeters** | **mils** | **inches** | **lambda** | **other**

Single-line comments beginning with // can be placed anywhere throughout the file. Curly brackets **{ }** delimit sets of items.

```
Layer = {
      LayerName="name"
      Lock=boolean
      Hidden=boolean
      AreaCapacitance=R
      FringeCapacitance=R
      Resistivity=R
```

```
CIFName="name"
GDSNum=N
ObjectPass = {
        SelectionPass = {
                ColorNumber=N
                WriteMode="mode"
                StipplePattern = {
                        H,H,H,H,H,H,H,H
                }
        }
        Pass1 = {
                ColorNumber=N
                WriteMode="mode"
                StipplePattern = {
                        H,H,H,H,H,H,H,H
                }
        }
}
PortPass = {
        SelectionPass = {
                ColorNumber=N
                WriteMode="mode"
                StipplePattern = {
                        H,H,H,H,H,H,H,H
                }
        }
        Pass1 = {
                ColorNumber=N
                WriteMode="mode"
```

```
            StipplePattern = {
                H,H,H,H,H,H,H,H
            }
        }
    }
    TextPass = {
        SelectionPass = {
            ColorNumber=N
            WriteMode="mode"
            StipplePattern = {
                H,H,H,H,H,H,H,H
            }
        }
        Pass1 = {
                ColorNumber=N
            WriteMode="mode"
            StipplePattern={
                H,H,H,H,H,H,H,H
            }
        }
    }
    Wire = {
        Width=I
        MiterAngle=I
        End="end"
        Join="join"
    }
}
```

```
SpecialLayer = {
        Grid="layer"
        Dragbox="layer"
        Origin="layer"
        CellOutline="layer"
        Error="layer"
        Icon="layer"
        FirstMask="layer"
}

Drawing = {
        DefaultPortTextSize=L
        NudgeAmount=N
        RulerSettings = {
                TextSize=N
                TextLocation="text"
                EndStyle="style"
                ShowTickMarks=boolean
                MajorTick=N
                MinorTick=N
                SymmetricTickMarks=boolean
                DefaultLayer="layer"
        }
}

Palette = {
        V6StylePalette=boolean
        RGBColorC=V,V,V
}
```

```
Technology = {
	Name="name"
	Unit_name="unit"
	Int_Unit_num=N
	Int_Unit_denom=N
	Lambda_num=N
	Lambda_denom=N
}

DerivedLayer = {
	TargetLayer="layer"
	EnableEvaluation=boolean
	SourceLayer1="layer",boolean,I
	SourceLayer2="layer",boolean,I
	SourceLayer3="layer",boolean,I
	Layer1BoolLayer2="operation"
	Layer2BoolLayer3="operation"
}

DesignRuleSetup = {
	RuleSet = "name"
	Tolerance = I
}

DesignRule = {
	RuleName="name"
	Enable=boolean
	RuleType="rule"
```

```
        IgnoreCoincidences=boolean
        IgnoreIntersections=boolean
        IgnoreEnclosures=boolean
        Ignore45AcuteAngles=boolean
        Layer1Name="name"
        Layer2Name="name"
        Distance=I
        UseLocatorUnits=boolean
}

Grid = {
        Displayed=U
        SuppressLessThan=P
        MouseSnap=U
        CursorType="cursor"
        LocatorUnit=U
}

Selection = {
        SelectionRange=N
        DeselectionRange =N
        EditRange = {
                Locator_Unit=N
                Pixels=N
        }
        DrawnObject="option"
}
```

Interpretation

Layer

An unlimited number of separate layer specifications may be made. The minimum requirement for a layer specification is the **LayerName**. The other parameters take default values as follows.

Parameter	*Default*	*More information*
Lock	FALSE	**Setup > Layers**
Hidden	FALSE	**Setup > Layers**
AreaCapacitance	0.0	**Setup > Layers**. Floating-point number.
FringeCapacitance	0.0	**Setup > Layers**. Floating-point number.
Resistivity	0.0	**Setup > Layers**. Floating-point number.
CIFName	null	**Setup > Layers** and CIF Files
GDSNum	null	**Setup > Layers** and GDS II Files
[minimum number of passes per pass list]	2	**Setup > Layers**

Parameter	*Default*	*More information*
ColorNumber	15	**Setup > Layers**
WriteMode	SET	**Setup > Layers**
StipplePattern	[empty]	**Setup > Layers**. **StipplePattern** is an 8x8 bit representation of the layer's stipple. Patterns are described as 8 pairs of hexadecimal numbers (for example: FF, 8B, A4) where each pair represents 8 bits. 8 such pairs represents 64 bits (8x8): the whole stipple pattern.
Width	0	Wire Styles
MiterAngle	90	Wire Styles
End	EXTEND	Wire Styles
Join	LAYOUT	Wire Styles

Special Layers

Only special layers to be modified should be named. The default values are as follows.

Parameter	***Default***	***More information***
Grid	Grid	**Setup > Special Layers**
DragBox	Drag Box	**Setup > Special Layers**
Origin	Origin	**Setup > Special Layers**
CellOutline	Cell Outline	**Setup > Special Layers**
Error	Error	**Setup > Special Layers**
Icon	Icon	**Setup > Special Layers**
FirstMask	Poly	**Setup > Special Layers**

Drawing

The default settings for the file and ruler parameters are as follows.

Parameter	***Default***	***More information***
DefaultPortTextSize	5	**Setup > Design**. Locator units.
NudgeAmount	1	**Setup > Design**
TextSize	5	**Setup > Design**
TextLocation	AT_TICK_MARKS	**Setup > Design**
EndStyle	NO_ARROWS	**Setup > Design**
ShowTickMarks	TRUE	**Setup > Design**
MajorTick	10	**Setup > Design**
MinorTick	1	**Setup > Design**
SymmetricTickMarks	FALSE	**Setup > Design**
DefaultLayer	Current Layer	**Setup > Design**

Palette

The color palette contains 16 different colors, with index numbers ranging from 0 to 15. The colors are made by mixing different amounts of red, blue, or green. The amount of each can be varied from 0 to 255. The defaults are as follows.

Parameter	***Default (red, blue, green)***	***More information***
V6StylePalette	TRUE	In previous versions of L-Edit, each color could be selected from one of 64 possible colors. In version 7, you can select each color from one of 16.7 million available. When a TTX file from a previous version is loaded into L-Edit, the **V6StylePalette** flag is inserted with the default value TRUE. This indicates that the earlier style palette should be used.
RGBColor0	255, 255, 255	**Setup > Palette**
RGBColor1	85, 85, 255	**Setup > Palette**
RGBColor2	85, 255, 35	**Setup > Palette**

Parameter	*Default (red, blue, green)*	*More information*
RGBColor3	85, 170, 170	**Setup > Palette**
RGBColor4	255, 85, 85	**Setup > Palette**
RGBColor5	170, 85, 170	**Setup > Palette**
RGBColor6	170, 170, 85	**Setup > Palette**
RGBColor7	170, 170, 170	**Setup > Palette**
RGBColor8	170, 170, 170	**Setup > Palette**
RGBColor9	0, 0, 170	**Setup > Palette**
RGBColor10	0, 170, 0	**Setup > Palette**
RGBColor11	0, 85, 85	**Setup > Palette**
RGBColor12	170, 0, 0	**Setup > Palette**
RGBColor13	85, 0, 85	**Setup > Palette**
RGBColor14	85, 85, 0	**Setup > Palette**
RGBColor15	0, 0, 0	**Setup > Palette**

Technology

The minimum requirement for a technology specification is the **Name**. The other parameters take default values as follows.

Parameter	*Default*	*More information*
Unit_name	microns	**Setup Design – Technology**
Int_Unit_num	1	**Setup Design – Technology**
Int_Unit_denom	1000	**Setup Design – Technology**
Lambda_num	1	**Setup Design – Technology**. Required only if **Unit_name** = “lambda.”
Lambda_denom	1	**Setup Design – Technology**. Required only if **Unit_name** = “lambda.”

Generated Layers

All required source layers should be defined according to **Layer = { … }** constructs before a generated layer is defined. There are no default values for generated layers.

Design Rules

A complete design rule specification requires a **DesignRuleSetup** definition and at least one **DesignRule** definition. If **UseLocatorUnits** is FALSE, then lambda units are used. There are no default values for design rules.

Grid

The defaults are as follows.

Parameter	***Default***	***More information***
Displayed	1	**Setup Design – Grid**
SuppressLessThan	8	**Setup Design – Grid**
MouseSnap	1	**Setup Design – Grid**
CursorType	SNAPPING	**Setup Design – Grid**
LocatorUnit	1	**Setup Design – Grid**

Selection

The defaults are as follows.

Parameter	*Default*	*More information*
SelectionRange	10	**Setup Design – Selection**
DeselectionRange	536870911 [maximum possible]	**Setup Design – Selection**
Locator_Unit	0	**Setup Design – Selection**
Pixels	2	**Setup Design – Selection**
DrawnObject	SELECT	**Setup Design – Selection**

XST Files

Syntax

A sample definition for an *n*-well, double-poly, double-metal CMOS process is shown below. Each line (after the header) corresponds to one process step.

```
# File: mORBn20.xst
# For: Cross-section process definition file
# Vendor: MOSIS:Orbit Semiconductor
# Technology: 2.0U N-Well (Lambda = 1.0um, Technology = SCNA)
# Technology Setup File: mORBn20.tdb
# Copyright (c)  1991-93
# Tanner Research, Inc.  All rights reserved
#
*****************************************************************
****
#       L-Edit
# Step Layer Name      Depth   Label  [Angle[offset]]  Comment
# ---------------------------------------------------------------
----
gd     -                 10    p-                # 1. Substrate
id     "Well X"          3     n-                # 2. n-Well
id     ActPSelNotPoly    0.9   p+     75    0    # 3. p-Implant
id     ActNSelNotPoly    0.9   n+     75    0    # 4. n-Implant
id     CCD&Act           0.4   -                 # 5. CCD Implant
id     "P Base"          2    -                # 6. NPN Base
Implant
gd     -                 0.6  -                # 7. Field Oxide
e      Active            0.6  -      45        # 8.
gd     -                 0.04 -                # 9. Gate Oxide
gd     Poly              0.4  -                # 10. Polysilicon
e      NotPoly           0.44 -      45        # 11.
gd     -                 0.07 -      45        # 12. 2nd Gate Oxide
gd     Poly2             0.4  -                # 13. 2nd
Polysilicon
e      NotPoly2          0.47 -      60        # 14.
gd     -                 0.9  -                # 15.
e      "P/P2/Act Contact"0.9  -      60        # 16.
gd     Metal1            0.6  -                # 17. Metal 1
e      "Not Metal1"      0.6  -      45        # 18.
gd     -                 1    -                # 19.
e      Via               1    -      60        # 20.
gd     Metal2            1.15 -                # 21. Metal 2
e      "Not Metal2"      1.15 -      45        # 22.
gd     -                 2    -                # 23. Overglass
e      Overglass         2    -                # 24.
```

Intrepretation

The Cross-Section Process Definition file (XST) contains a list of comment statements and process statements. No blank lines are allowed in the file; processing stops at the first blank line. Comment statements begin with a pound sign (#) in the first column and continue to the end of the line.

Process statements have the following format:

```
step layer depth label [angle [offset]] [comment]
```

The line begins with a **step** type, one of the following:

- **gd** or **grow/deposit**
- **e** or **etch**
- **id** or **implant/diffuse**

Next is the name of the involved **layer**. The name of the layer must match the layer name used in the L-Edit TDB file. If the layer name begins with a digit or contains spaces, then the entire name must be enclosed in double-quotes (**"**…**"**). The layer name describes something different for each type of step:

- For grow/deposit steps: the layer to be grown/deposited
- For etch steps: the layer to be etched away
- For implant/diffuse steps: the layer to be diffused

A dash (**-**) in place of a layer name indicates that the process step has no associated rendering information.

Next is a (non-negative) value indicating the **depth**, measured in technology units. The depth also means different things for different steps:

- For grow/deposit steps: the number of units to grow upward
- For etch and implant/diffuse steps: the number of units downward to apply the step

Next is an optional **label**. The label may be any string. If it contains spaces, then the entire label must be enclosed in double-quotes (**"**…**"**). A dash (**-**) may be used in place of a label.

If desired, two parameters that apply only to etch and implant/diffuse steps are inserted next:

- Etch-implant **angle** (integer)
- Undercut **offset** (non-negative floating-point or integer)

Angles are measured in degrees and must be between 0 and 180; offsets are measured in technology units. The default values are **angle** = 80 and **offset** = 0.

Last is an optional **comment**. The comment begins with a pound sign (**#**) and continues to the end of the line.