cadence®

Transferring Logic Design Data

Product Version 23.1
September 2023

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

1

Logic Transfer Methodology

This section and others describe how to transfer native and third-party design logic data to the back-end layout editors and backannotate that data. Native logic transfer refers to data derived from Allegro Design Entry HDL, System Connectivity Manager, or Allegro PCB Design CIS, Cadence front-end tools that create schematics.

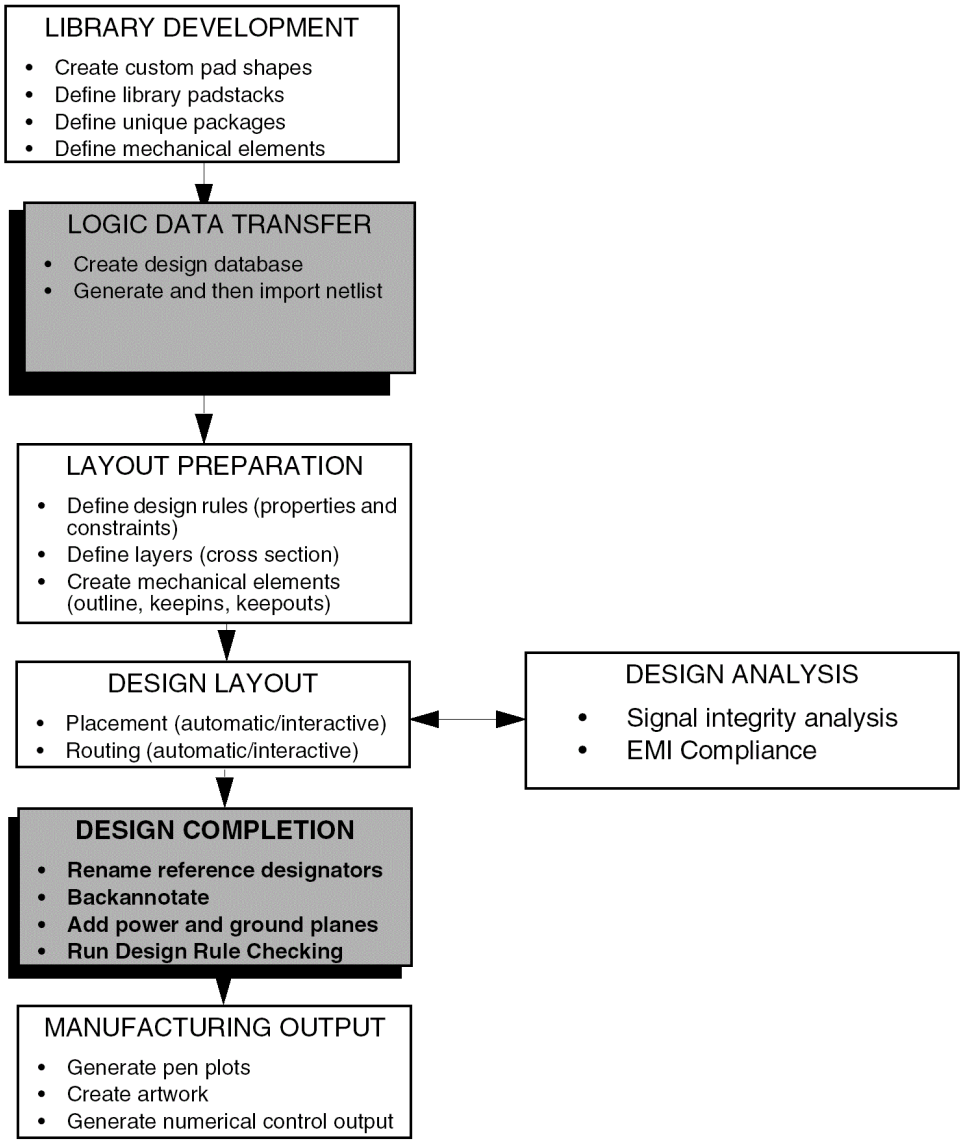
⚠ Many features are common to the layout editors: Allegro X PCB Editor and Allegro X Advanced Package Designer. When a feature is not common to all editors, it is noted in the heading. If an illustration shows only one of the editors, it is also noted.

The information in this section describes functionality available in the layout editor from the design perspective. Step-by-step procedures for performing these functions are available in the *Allegro PCB and Package Physical Layout Command Reference*.

The methodology is based on Project Manager, Cadence's flow control tool; Allegro Design Entry HDL, System Connectivity Manager, or OrCAD PCB Designer; and Packager-XL – the interface between the logic design and the physical layout. The functionality of these tools is not documented in this user guide.

Logic transfer is performed (*Import/Export Logic* or *Export/Import Physical*) to send or synchronize data between the schematic and the layout editor. This synchronization (forward and back annotation) usually occurs frequently throughout the design process. The following figure shows this process.

Logic Transfer in a Design Flow



2

Transferring Logic

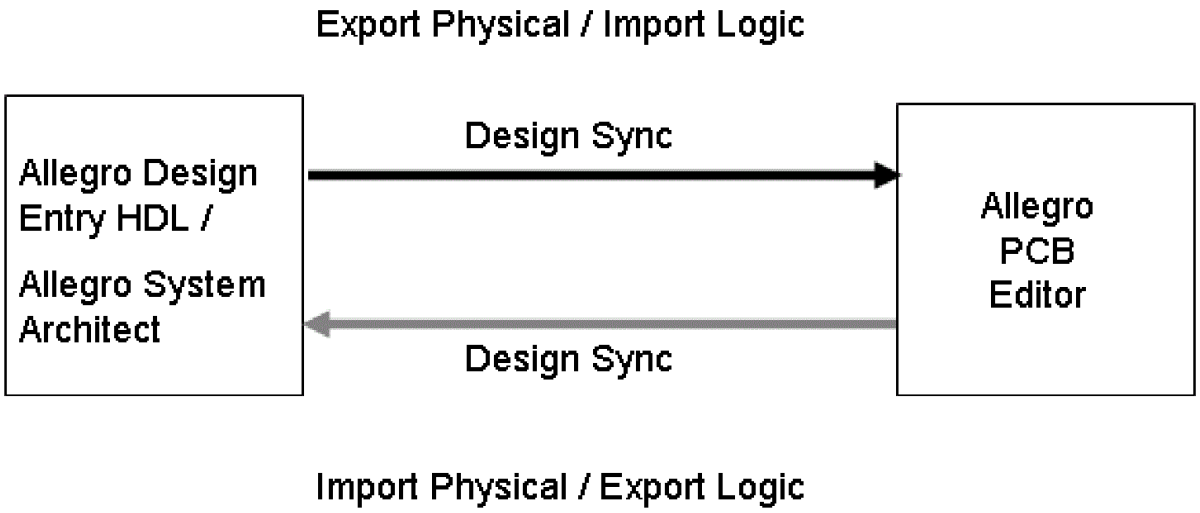
⚠ Although this feature is available in Allegro Package Designer+, it is rarely used.

Native design logic refers to design data extracted from Allegro Design Entry HDL, System Connectivity Manager, or OrCAD X PCB Designer, Cadence front-end products that create schematics or logic design files and which interface with the layout editor. You transfer design logic between them using a process called design synchronization (Design Sync) to import attributes (properties or constraints) attached to pins, functions, modules, parts and nets. However, you cannot transfer pin and function attributes from OrCAD X PCB Designer.

⚠ An attribute in Allegro Design Entry HDL and System Connectivity Manager, or a property in the layout editor, takes precedence over an ECset value. In Constraint Manager, a property on a net appears as a net override with the value of the property rather than the ECset value. Differential pair values have higher precedence than properties.

The following figure provides an overview of logic transfer from Allegro Design Entry HDL or System Connectivity Manager to the layout editor and back again. Transfer of logic from OrCAD X Capture CIS is similar to logic transfer from Allegro Design Entry HDL or System Connectivity Manager. OrCAD X Capture CIS does generate the `pst.*dat` files but without the constraint files, as it does not have access to Constraint Manager.

Example: Allegro Design Entry HDL or System Connectivity Manager to PCB Editor



The transfer of logic involves performing tasks in Cadence's front-end and back-end products to insure that changes between the logic design data and the physical design update correctly.

HDL

Cadence's front-end schematic capture products, Allegro Design Entry HDL and System Connectivity Manager, support the HDL logic import or export modes.

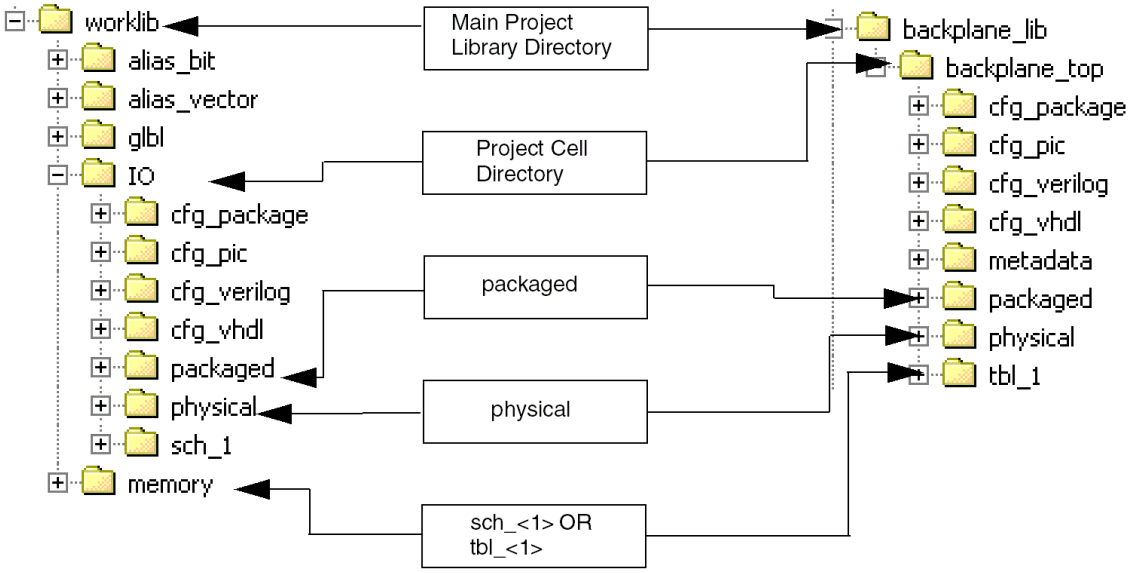
Packaging design logic generates the file type (`pst*.dat`) in HDL.

Allegro Design Entry HDL and System Connectivity Manager create a directory structure that places all files of a project (logical, physical, and so on) under their associated cell (design name). After Design Sync, packaged files appear in a subdirectory named *packaged* within this hierarchy, as follows.

HDL-Based Transfer Files Directory Structure

Allegro Design Entry HDL Schematic Files

System Connectivity Manager Design Files



Conversion Restrictions

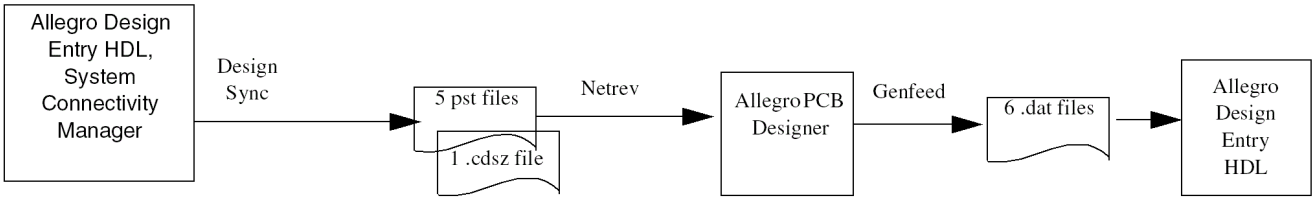
When you import or export logic, the layout editor brands the database as Allegro Design Entry HDL or System Connectivity Manager, restricting transfer files to files of the same type or third party. For example, an Allegro Design Entry HDL or System Connectivity Manager database can read Allegro Design Entry HDL or System Connectivity Manager or third-party transfer files, but not OrCAD X PCB Designer transfer files.

Logic Transfer Flows

The number of package files that pass from Allegro Design Entry HDL to the layout editor depends on how you defined the electrical constraints. Allegro Design Entry HDL generates additional package files when you transfer logic if you use Constraint Manager. The additional files, `pstcmdb.dat` and `pstcmbc.dat`, contain constraint data that Constraint Manager needs in the layout editor. System Connectivity Manager, however, transfers only one `.cdsz` file.

There is no support for constraint files (`pstcmdb.dat` and `pstcmbc.dat`) in third-party schematic flows.

PCB Editor: Front-to-Back Flow Diagram



⚠ System Connectivity Manager generates one `.cdsz` file.

Related Topics

- Transferring Logic Created by a Third Party
- Transferring Logic to the Layout Editor

Transferring Logic to the Layout Editor

This section describes design synchronization tasks for transferring design data from a Allegro Design Entry HDL `pst*.dat` file or an System Connectivity Manager `.cdsz` to the physical design.

Extracting Logic from Allegro Design Entry HDL or System Connectivity Manager

Run Design Sync from Allegro Design Entry HDL, System Connectivity Manager, or Project Manager to convert the logic devices into physical packages and assign reference designators and physical pin numbers to each symbol in the Allegro Design Entry HDL schematic or System Connectivity Manager design files. The packaged parts and their connections write into `pst*.dat` files or a `.cdsz` file that transfers information from the schematic to the physical design.

Preparing to Import Design Data

You cannot create a physical design by transferring design logic to the back-end tools. When importing logic into a layout editor, you update an existing design. If you do not have a starting design, you can create one before importing logic, as follows:

1. Start the layout editor and open a design.
2. Set up the cross section and the board outline (PCB Editor).
3. Define user properties in the layout editor that correspond to the Allegro Design Entry HDL or System Connectivity Manager user properties.
4. Make sure that the `devpath` environment variable is set to the location of the `pst*.dat` files or `.cdsz` file.

Type set at the console window prompt to display the *Defined Variables* window and view the `devpath` string. To set or edit this variable, manually edit the `/pcbenv/env` file or use *Setup – User Preferences* (`enved` command).

Verify settings in the *User Preferences* dialog box.

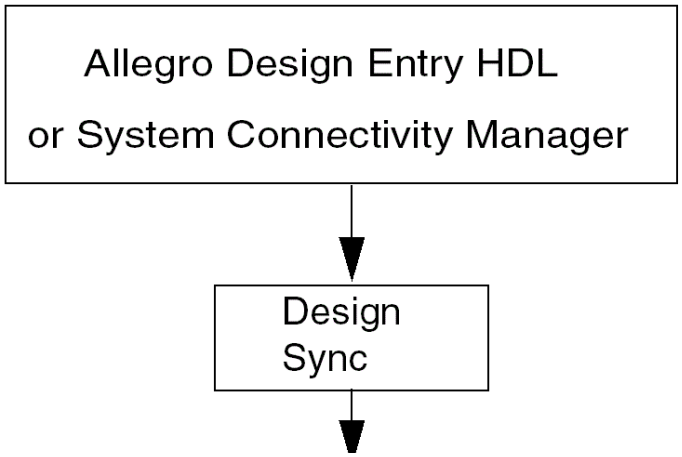
1. Choose *File – Save* (`save` command) or *File – Save As* (`save_as` command), where appropriate.

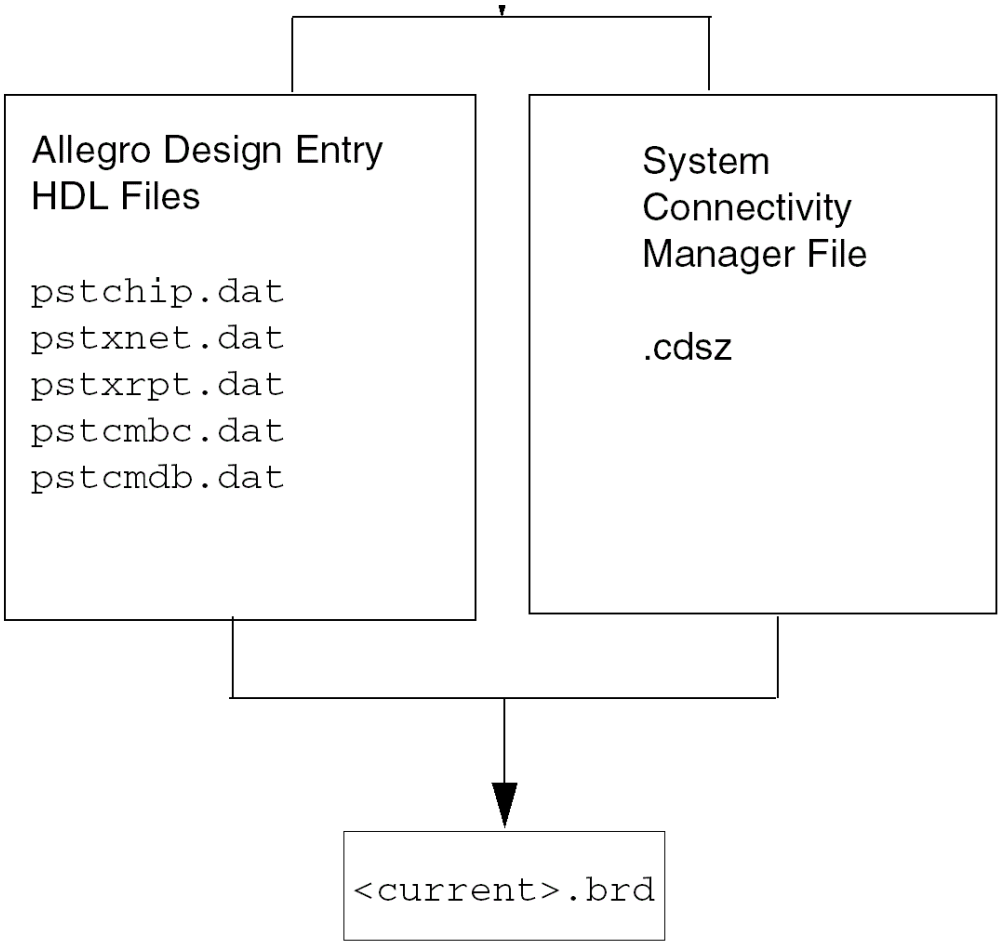
⚠ You can also perform some or all of these steps after importing logic.

Updating or Importing Design Data

To update the layout, use the `netrev` utility, which reads the `pst*.dat` or `.cdsz` transfer files into a physical design. You can run this utility either through the *Import Logic* dialog box, that displays when you choose *File – Import – Logic*, or by running the `netrev` batch command. The following figure shows the packaged files generated during Design Sync.

PCB Editor: Design Sync Files – Forward Mode





Note: You must have all .*psm and .*pad files in the location specified in the psmpath and padpath. By default, the layout editor first looks for these files in the working directory.

The netrev utility:

- Imports the packaged logic data.
- Searches the library in the layout editor working directory for package symbols and alternate symbols specified in the pstchip.dat file. If not found, the layout editor looks for them in the design path locations, such as, padpath, psmpath, or modulepath.

If unable to locate a package or padstack, `netrev` generates a warning message for each missing device and symbol package, or pad. If module files (`.mdd` for reuse) are missing or a module path is incorrect, no errors or warnings appear in `netrev.lst`. When you place the modules, however, an error occurs stating that the module definition is not found. Correcting the path or supplying the files remedies the problem.

- Compares package symbols found in the library to the device definitions in `pstchip.dat`. Mismatches generate error messages.
- Creates a log file, `netrev.lst`, that indicates whether the process was successful and lists warnings and errors. (`netrev.lst` is appended to the `concept.log` file)
- Generates a change summary report, `eco.txt`

> ⚠ The console window prompt displays status on the *Import Logic* process, including the number of errors or warnings in the log file.

Import Logic I/O Files

The `netrev` utility inputs the following files:

`<package>.psm`	A package or part symbol file. In most runs, *Import Logic* accesses several <package>.psm files. Package or part symbol files are graphical representations of electronic devices such as dual in-line packages (DIPs), connectors, resistors, and capacitors. The name of the package or part symbol file comes from the `pstchip.dat` file; the layout editor appends the `.psm` extension.
`pstchip.dat`	Device definition file that contains physical information for each type of symbol read from the `chips.prt` files and the physical parts tables, including electrical characteristics such as pin direction and loading, logical to physical pin mapping, and voltage requirements. It defines the number of gates in each device, including gate and pin swapping information. Also contains the name of the package or part symbol used to represent the device type in the physical layout (JEDEC_TYPE). Device files are the third party equivalent of this file. This file is output from the Packager.

`pstxnet.dat`	Netlist file that uses keywords (`net_name, node_name`) to specify the reference designators and pin numbers associated with each net in the schematic. The file contains attributes or properties attached to nets in the schematic. Constraints added to nets using Constraint Manager are found in `pstcmdb.dat`. This file is output from the Packager.
`pstxprt.dat`	The gates in the design by the reference designator. Lists each physical package or part in the schematic along with its reference designator and device type. For packages or parts composed of multiple logic gates, the file identifies which gate was placed in which section of the package or part. Also contains attributes for parts, functions, and pins unless they were added in Constraint Manager, in which case they are in `pstcmdb.dat`. This file is output from the Packager.
`pstcmdb.dat`	Definitions of electrical constraints in the schematic as defined and created in the Constraint Manager database. These files must be present for `netrev` to import complex electrical constraints.
`pstcmbc.dat`	The electrical constraint baseline in the schematic data.
`*.pad`	A padstack definition defined in the layout editor library or a user-defined library.

`netrev` outputs the following files:

`netrev.lst`	Log file containing errors and other problems encountered by *Import Logic* when loading design data.
`eco.txt`	Log file containing all changes to a database that result from loading logic design data. Changes include the change of loading logic design data for the first time into the database.
`current.brd`	The design data loaded into the current.

⚠ The `netrev` command does not check mechanical files (`.bsm`), format symbols (`.osm`), shape symbols (`.ssm`), module (`.mdd`), and flash symbols (`.fsm`).

Allegro Design Entry HDL and System Connectivity Manager Properties That Import into the Layout Editor

Allegro Design Entry HDL and System Connectivity Manager properties that import into the layout editor:

- Directly map to corresponding editor elements: `GROUP` and `ROOM`.
- Map to different editor elements: `LOCATION` and `HAS_FIXED_SIZE`.
- Affect the physical design only: `PACK_TYPE`.
- Are affected by constraint sets: `ELECTRICAL_CONSTRAINT_SET`.
- Are user-defined properties.

Property Deletion or Modification

If you add or modify a property in Allegro Design Entry HDL or System Connectivity Manager, the layout editor adds or modifies the property during logic transfer even if you have modified the property in the editor. If you delete a property in Allegro Design Entry HDL or System Connectivity Manager, the layout editor deletes the property during logic transfer only if you have not modified the property in the editor.

⚠ Migrating designs created before Release 15.0 requires that you run *File – Import – Logic (`netrev` command)* once before you can delete properties on subsequent `netrev` operations.

Allegro Design Entry HDL or System Connectivity Manager Properties Not for Use with the Layout Editor

The `NO_SWAP_GATE` and `NO_SWAP_EXT` properties are designed to work with third-party software and create problems when `netrev` tries to load them into the layout database. Do not attach these properties to parts in the Allegro Design Entry HDL schematic or System Connectivity Manager design files.

Related Topics

- `netin`
- `netrev`
- Allegro X Platform Properties Reference

Backannotating to Allegro Design Entry HDL or System Connectivity Manager

When you swap gates, change properties and constraints, rename reference designators and execute netlist-driven engineering change orders (ECOs) to a layout (that cause it to become logically out of synchronization with its associated schematic), you need to communicate those changes back to the schematic. This process is called backannotation.

Backannotating documents changes to reference designators and physical pin numbers, as well as, properties specified by `pxlBA.txt`. To perform properly, the design logic and physical layout must match. If parts exist in the schematic that are not in the design (or vice versa) or if schematic connectivity does not match the physical layout, the layout editor identifies these differences.

If you use logic/net logic to create, rename, or remove nets and assign or unassign pins to them, these changes cannot backannotate to the schematic or logic design files. In System Connectivity Manager, only those properties specified in the *Setup* dialog box in the *Property Flow* section are also chosen in the backannotate.

Creating the Output Files

Backannotation with the layout editor runs the extract program that creates a temporary file from the active design and from properties specified in the `pxlBA.txt` file. The layout editor then creates the required *view.dat files that include:

- `compview.dat`
  Component properties that have changed in the layout editor.
- `funcview.dat`

Function properties that have changed in the layout editor.

- `netview.dat`
  Net properties that have changed in the layout editor.
- `pinview.dat`
  Pin properties that have changed in the layout editor.
- `cmdbview.dat`
  Electrical constraints that have changed in the layout editor.
- `cmbcview.dat`
  Baseline constraints.

⚠ The layout editor creates only one `feedbackview.cdsz` file when you backannotate to System Connectivity Manager.

Generating Feedback Files from Physical Designs

You can generate feedback files by running the batch command, `genfeedformat`, from the operating system prompt.

Updating the Changes in the Design

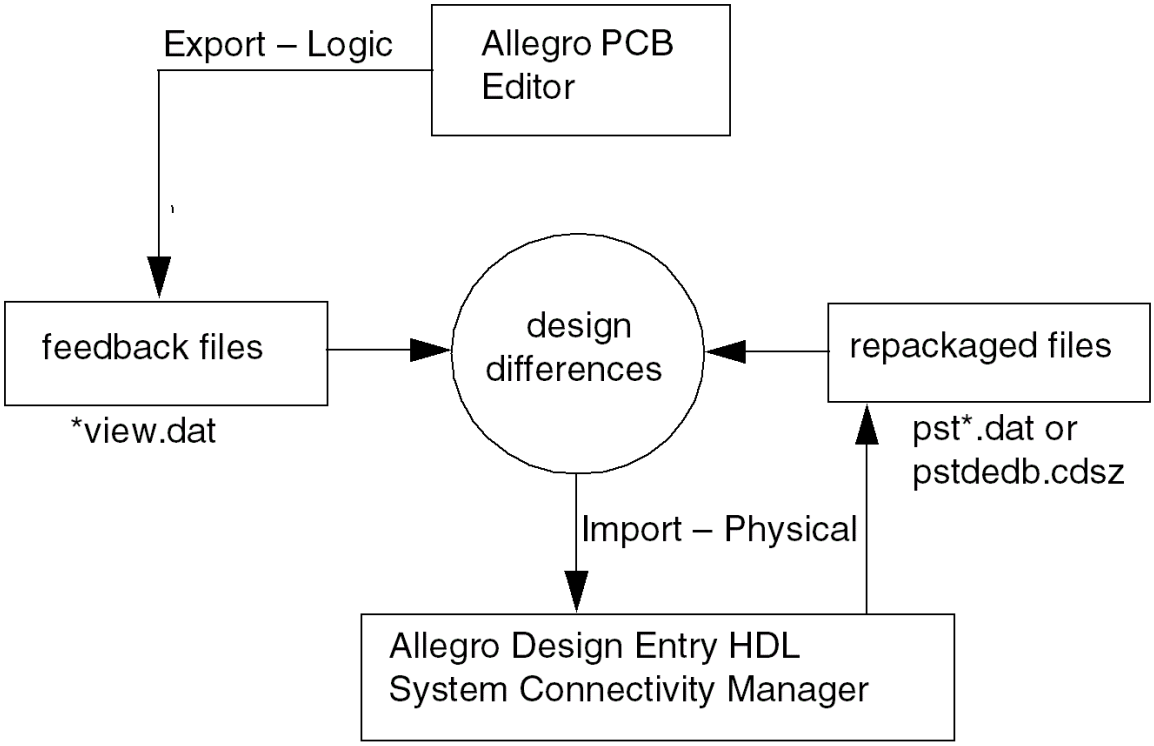
In Allegro Design Entry HDL, run Design Sync in backannotation mode to repackage the schematic for updating. This process generates the following files:

- `pstback.dat`
  The backannotation file produced whenever the Packager executes. Allegro Design Entry HDL uses this file to update the schematic.
- `pxl.chg`
  This file lists any differences discovered by the Packager between the intermediate files and the schematic.
- `pstcmback.dat`
  This file backannotates the changes in constraints in the design to the schematic.

> ⓘ All design changes backannotate when using Allegro Design Entry HDL.

In System Connectivity Manager, choose *File – Import Physical* to repackage the schematic for updating the logic design files. You can choose to update all changes in the physical design to the logical design or update the changes from the previously generated feedback files to the logical design. System Connectivity Manager lets you examine the differences in the feedback files and choose whether to update all or only individual changes to the logical design. The following figure shows the files involved in backannotating logic.

PCB Editor: Design Sync in Backannotation Mode



Related Topics

- feedback
- genfeedformat

3

Transferring Logic Created by a Third Party

This section describes how to transfer logic data from third-party designs to the layout editor and backannotate data to the schematic.

⚠ Although this feature is available in Allegro X Advanced Package Designer, it is rarely used in these products.

Third-party logic transfer refers to any schematic information derived from non-Cadence sources. This includes netlists and device files you may have created independently.

Netlists contain component, property, and connectivity data. Device files contain physical information for the components in the netlist. One device file is required for each device type. The netlist is read into a physical design using the `netin` command which is accessed by choosing *File – Import – Logic* (`netin` command). A log file `(netin.log)` lists any errors found in the netlist and device files.

Many third-party products can output a netlist formatted for the physical design.

You can also create netlists and device files using a text editor such as Notepad on Windows or vi in UNIX.

Third-Party to Layout Editor Process Flow

1. The third-party tool creates netlist and device files based on the schematic.
2. Choose *File – Import – Logic* (`netin` command) to import the schematic logic into the active physical design.
3. Use the layout editor to place, route, and run high-speed analysis with SigXplorer.
4. *File – Import – Logic* (`netin` command), *File – Export – Logic* (`feedback` command), or *genfeedformat* to backannotate changes to the schematic. Import/export logic may be performed as often as necessary.
5. Use the layout editor to generate manufacturing output.

Related Topics

- Transferring Logic to a Layout Editor
- Backannotating Third-Party Data
- netin
- feedback
- genfeedformat
- Defining and Developing Libraries

Transferring Logic to a Layout Editor

This section outlines transferring design logic extracted from a third-party source to a physical design. The imported logic data comprises the database for the layout.

Use the NET_SPACING_TYPE property for a spacing Net Class and the NET_PHYSICAL_TYPE property for a physical Net Class. These netlist properties are mapped in PCB Editor to Net Class groups. The syntax used to specify these properties follows the third-party netlist format.

For example, you can assign a spacing Net Class VOLTAGE to GND and VCC nets using the following format:

```
$A_PROPERTIES NET_SPACING_TYPE 'VOLTAGE';'GND' 'VCC' Caution:
```

The third-party netlist suffix the name of the physical Net Class with a "_PH" if a Net Class name already exists in the spacing domain.

Netin.log

The following example shows the contents of a netin.log file after netin has checked syntax. In this example, a comma was omitted from the a list of reference designators that continued on another line. netin assumed that the reference designator, *Y29*, is a package or part name.

```
(NETLIST)
(FOR DRAWING: /allegro_test/dfa.brd)
(Thu Nov 19 12:28:54 1995)
$PACKAGES
CAPCK05 ! 'CAPACITOR-1' ; C2 C4 C6 C8
CAPCK05 ! 'CAPACITOR-2' ; C1 C3 C5 C7
```

```
CONN10 ! CONNECTOR ; J1 J2 J3
DIP14 ! 74F02 ; N20 N29
DIP14 ! 74F74 ; N05 N08 N11 N14 N17 N23 T11
Y29
^
ERROR: Expected '!' before device, line ignored.
```

⚠ If you do not define user properties in the layout editor before importing netlist you will get the following warning message in netin.log file.

```
WARNING(SPMHNI-290): Property '%s' is not defined.
```

Backannotating Third-Party Data

When you swap gates and pins, rename reference designators, and execute netlist-driven ECOs to a physical design that causes it to become logically out of synchronization with its associated schematic, you need to communicate those changes back to the schematic. This process is called *backannotation*.

Backannotation returns reference designator and physical pin number changes only. To perform properly, the schematic and physical layout must match. If parts exist in the schematic that are not in the physical design (or vice versa) or if schematic connectivity does not match the physical layout, these difference are identified. Therefore, avoid modifying the schematic, parts, or device files generated by the schematic unless you import those changes to the layout editor before performing backannotation.

This section details the procedure for backannotating information to third-party tools.

Creating the Output File

Choose *File – Export – Logic* (`feedback` command) to create a `filename.baf` file from the active design. This file contains reference designator assignments (after gate/pin swap, or reference designator rename), as well as a was/is list of all pins for each part, indicating changes that may have occurred during gate and pin swapping.

Related Topics

- feedback
- Exporting Third-party (Other) Logic

4

Converting Third-Party Designs and Mechanical Data

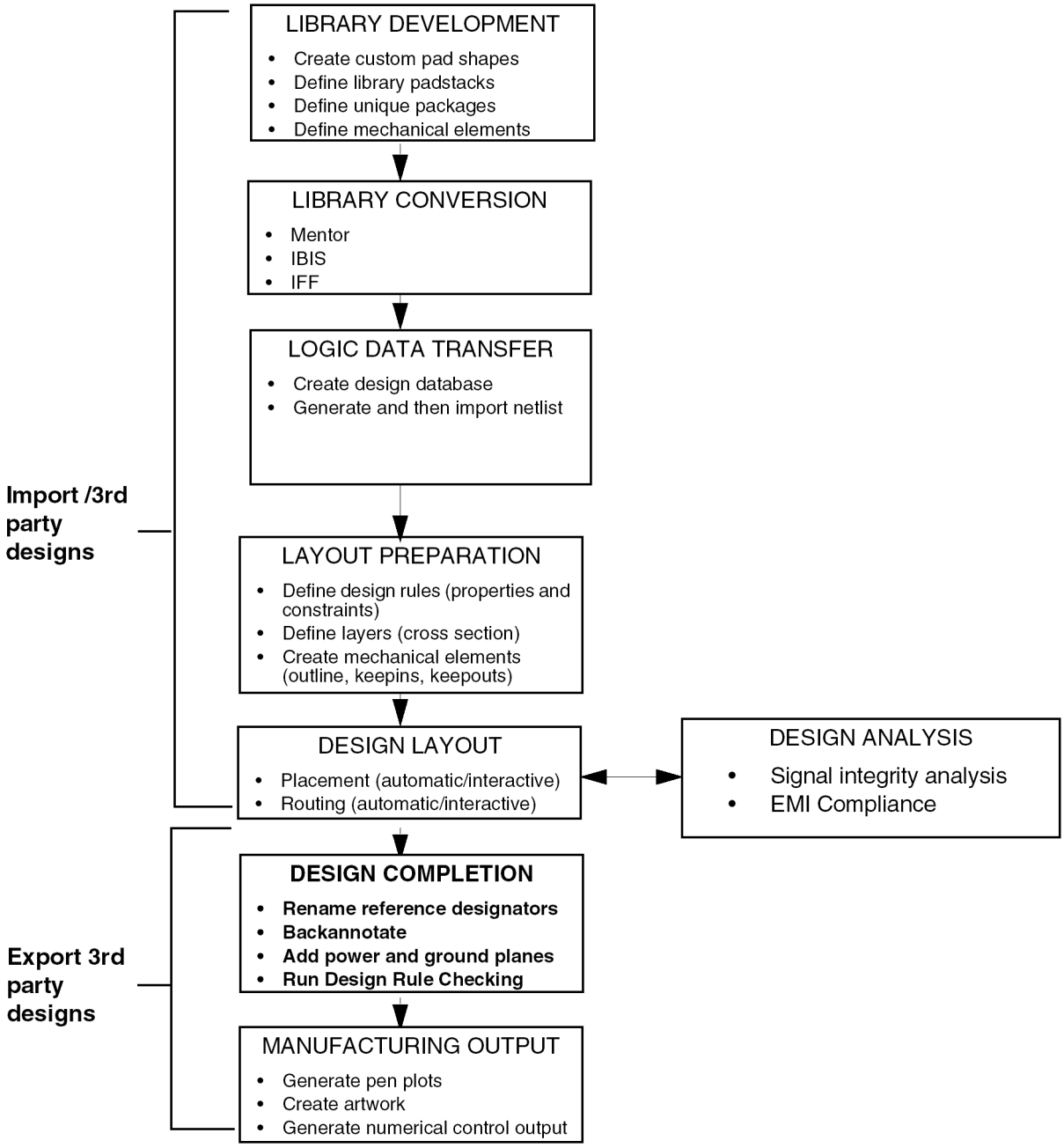
The layout editor provides commands that convert design data from the following Electrical Design Automation (EDA) format into the physical designs:

EDA File Format	Keyboard Command	Layout Editor Menu Selection
DXF	`dxf in/dxf out`	*File – Import/Export – DXF*
IDF PT	`idf in/idf out`	*File – Import/Export – IDF*
IDX	idx in/ idx out	File – *Import/Export – IDX*
IF	`iff in`	*File – Import – IFF*
IPC 356	`IPC356_out`	*File – Export – IPC356*
IPC 2581	IPC 2581 compare/ IPC 2581 in/ IPC 2581 out	File – *Export/Import – IPC 2581*
PowerPCB or Pads Layout	`pads in`	*File – Import – PADS*
PCAD	`pcad in`	*File – Import – PCAD*
GDSII	`load stream stream out`	*File – Import – Stream Manufacture – Stream Out*
NA2 (EncoreBGA)	na2 import	*File – Import – SPD2/NA2*
Valor ODB++	`odb_out`	*File – Export – ODB++*
Mentor-to-Allegro X PCB Editor Translator	`mbs2brd`	Operating System Prompt

Mentor-to-Allegro X PCB Editor Library Translator	`mbs2lib`	Operating System Prompt
SPD2 (UPD)	na2 import	*File – Import – SPD2/NA2*

Design conversion can occur at different stages in the design flow, depending on whether you are importing or exporting data. Normally, data is imported early in a flow and exported near or after completion.

Third-Party Conversion in a Design Flow



Optimizing the Use of Layout Editor Converters

To get the most from the converters, you must evaluate how they operate and how they plan to operate in the future. Whenever new systems are developed, changes to the design process are inevitable. This is the time when you can make choices on where you want to proceed in the future. The following scenarios can help you define a process. Every site is different, however, and not every scenario is applicable.

When converting designs, you must make several choices, deciding when and which designs will be converted. You must decide if you want to use the master libraries you already converted, or the library created with each design.

Run several conversions to learn and define the process. This lets you build a checklist that can be used by anyone at your site who will also be converting designs. The conversion process ensures:

1. Running the conversion command (`pcad in, dxf in, ...`).
2. Cleaning up and inspecting manually.

Running Conversion Commands

Each converter operates similarly, but there are some differences. You should become completely familiar with the documentation for the converter you are using.

To display the dialog boxes associated with each converter:

From the menu bar, choose

- *File – Import – <EDA tool>*
  OR run the command for that tool

Manually Cleaning up

Manual cleanup is the next step in the conversion process. In manual cleanup, do the following:

- Autovoid any shapes that require autovoiding.
- Realign reference designators and add any other text that did not convert.
- Scan the design and review all design rule markers.
- Review and, if necessary, modify padstack data, text block information, and summary reports.
- Connect shapes to appropriate nets.

- Derive connectivity, if necessary.
- Set up for photoplotting.

Inspecting the Design

You should treat the design as if it were a brand new design and thoroughly inspect it. Spending time inspecting could save you costly rework on an incorrectly manufactured design.

DXF Bi-Directional Interface

The Drawing Interchange Format (DXF) interface lets you exchange graphical data from a layout design with that of other mechanical computer-aided design (CAD) systems. You can import layout editor-supported DXF entities from DXF files written according to R10 DXF to AutoCAD 2000 DXF specifications or export mechanical design data to a DXF file according to DXF Revision 12 or Revision 14 specifications.

Reading DXF Files from AutoCAD into the Layout Editor

To read your DXF file into the layout editor, you must set certain parameters in your DXF files to ensure that the layout editor imports the data correctly.

DXF files originating from AutoCAD must contain these blocks and attributes if the layout editor is to correctly read the DXF files. Symbols and vias in a DXF file are mapped as blocks. A DXF block with a `CDN_TYPE` attribute associated with it is considered to contain the layout editor specific data. The layout editor ignores any block without this attribute during translation. This distinction is necessary as AutoCAD now generates some generic blocks by default.

⚠ In defining a `CDN_TYPE` attribute, you must enter a default at the `.dwg` level. Once properly defined, the `.dwg` is then brought out to a `.dxf` format, that is then read into the layout editor.

Via Representation in a DXF file

In a DXF file, a PADSTACK block with the `CDN_TYPE` attribute set to `VIA` represents a via in the layout editor. A `VIA` block can only be present in the `ENTITY` section of the DXF file or in a `SYMBOL` block. The layout editor ignores a `VIA` block inside any other block, and a `PIN` attribute inside a `VIA` block.

Pin Representation in a DXF File

In a DXF file, a `PADSTACK` block with the `CDN_TYPE` attribute set to `PIN` represents a pin in the layout editor. A `PIN` block can exist in the `ENTITY` section of the DXF file only when importing DXF in a symbol drawing, or in a `SYMBOL` block.

Symbol Representation in a DXF File

In a DXF file, a block with the CDN_TYPE attribute set to SYMBOL represents a symbol from the layout editor. The block name is the same as the symbol name. A SYMBOL block has PIN blocks to represent pins for this symbol. The positioning and rotation of these PIN blocks determines pin rotation and position.

A SYMBOL block can only be present in the ENTITY section of the DXF file. It is ignored inside any other block. A SYMBOL block can also have other geometric entities on different DXF layers to represent symbol data in the layout editor.

Symbol and Padstack Definition Support

The dxf2a program generates symbol and padstack definitions from DXF blocks that adhere to the desired format as follows.

DRILL_INFO block

A DRILL_INFO block must contain the seven attribute definitions listed below, defining the necessary drill information on each layer for this padstack:

Attribute Tag	Fixed Value	Variable Value
CDN_TYPE	DRILL_INFO	
DIAMETER		Diameter of drill hole in decimal format
PLATED	Drill plating	0 or PLATED 1 or NON_PLATED 2 or OPT_PLATED
FSHAPE	DRILL shape	CIRCLE CROSS DIAMOND HEXAGON_X HEXAGON_Y OBLONG OCTAGON RECTANGLE SQUARE TRIANGLE
FCHAR	DRILL character	Any single keyboard character, including symbols such as #, %, and so on
DRILL_WIDTH		Drill figure width measured along x-axis in decimal format
DRILL_HEIGHT		Drill figure height measured along y-axis in decimal format

PAD_INFO Block

A PAD_INFO block must contain the six attribute definitions listed below. Each PAD_INFO block corresponds to pad information on each ETCH/CONDUCTOR layer and contains necessary pad-related information using attributes. The PAD_INFO block has following attributes defining the pad information on each layer:

Attribute Tag	Fixed Value	Variable Value
CDN_TYPE	PAD_INFO	
PAD_LAYER	DXF layer name	Any keyboard character, including symbols such as #, %, and so on.
SHAPE_TYPE	Pad shape	1 or CIRCLE 2 or SQUARE 3 or OBLONG 4 or RECTANGLE
PAD_TYPE		0 or PAD_REGULAR 1 or ANTIPAD 2 or THERMAL
PAD_WIDTH		Width of pad in decimal format
PAD_HEIGHT		Height of pad in decimal format

PADSTACK Block

The block name is the padstack name, which in turn instantiates blocks to represent padstack data. The PADSTACK block also has multiple instances of blocks with the CDN_TYPE attribute set to PAD_INFO. A padstack block must contain the four attribute definitions listed below, as well as one or more instances of a PAD_INFO block and none or one instance of a DRILL_INFO block. Any geometry included with the block is ignored. The PADSTACK block has the following attributes:

Attribute Tag	Fixed Value	Variable Value
CDN_TYPE		VIA or PIN
PSTKNAME	Padstack name (256 character limit). Standard editor naming rules apply.	

BBV	THRU or BLIND	
PIN		Pin number

SYMBOL Block

A SYMBOL block must contain the three attribute definitions listed below in addition to its geometric entities.

Attribute Tag	Fixed Value	Variable Value
CDN_TYPE	SYMBOL	
SYMBOL_TYPE	PACKAGE, MECHANICAL FORMAT	
REFDES		text

ⓘ The layout editor ignores any other entity present in DRILL_INFO, PAD_INFO, and PADSTACK blocks.

You may want placeholder entities in these blocks to visually represent them in the AutoCAD system. The layout editor does not generate any other entity in DRILL_INFO and PAD_INFO blocks. DRILL_INFO and PAD_INFO blocks can only be present inside a PADSTACK block. The editor ignores them anywhere else.

Creating a DXF File from a Physical Design

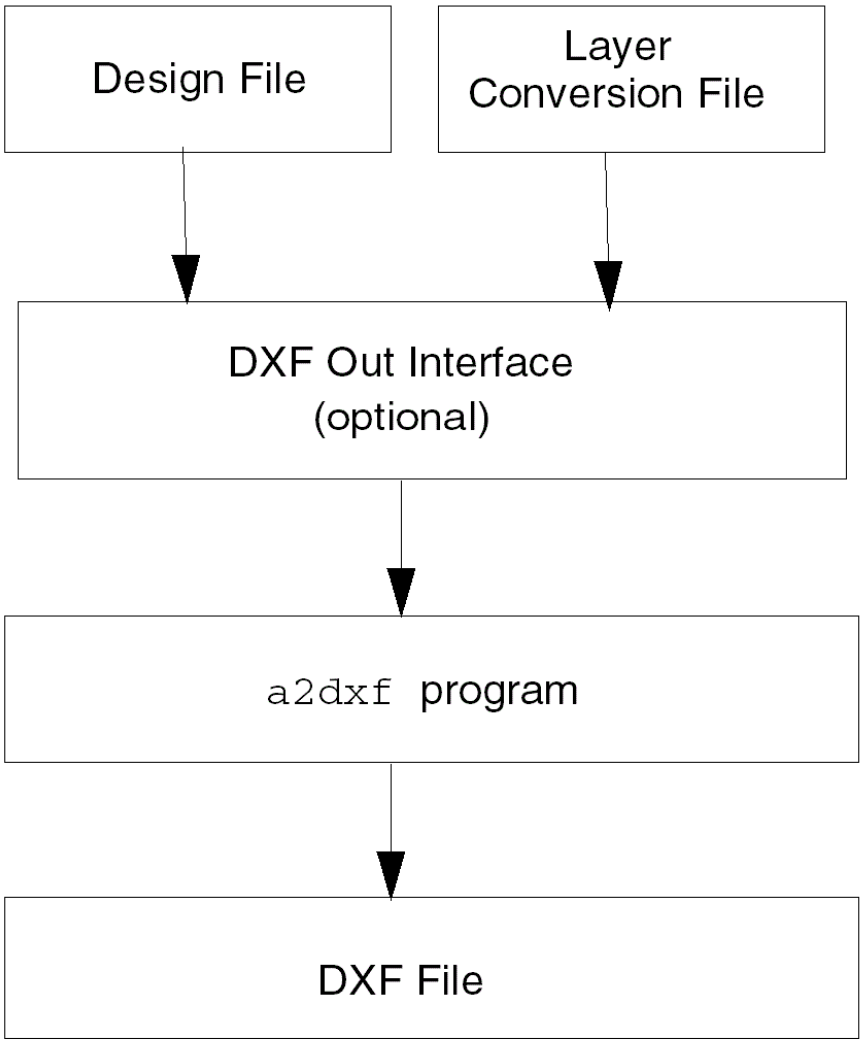
You can export mechanical design data to a DXF file using the a2dxf program, executed by choosing *File – Export – DXF* (dxf out command).

The a2dxf program exports mechanical design data from a design file and creates a DXF file in ASCII format. You can also use the a2dxf program to selectively output certain classes and subclasses that correspond to specific layers in a DXF file. Additionally, the a2dxf program:

- Automatically creates a default layer conversion file, if the layer conversion file you specified does not exist
- Converts the mechanical design data from the physical design into a DXF file using:

  - The DXF Out dialog box parameters that you specify
  - A layer conversion file (either one that you specify or the default file)
- Displays any processing and status messages on the screen and creates an `a2dxf.log` file that contains the informational messages.

Layout Editor to DXF Process



Using the DXF Out Dialog Box

When you run the `a2dxf` program using *File – Export – DX*F (`dxf out` command), the DXF Out dialog box appears, which lets you specify:

- DXF File Format: Revision 12 or 14
- The level of decimal precision (accuracy) of numeric values in the DXF file
- Whether symbols and padstacks should be exported as blocks to the DXF file
- Whether to export filled pads
- Whether to fill solid shapes
- Whether to export drill information to the DXF file
- The name of a layer conversion file that identifies the classes and subclasses corresponding to the DXF layers to be contained in the `.dxf` file

Layout Editor to DXF Entity Mapping

The layout editor `a2dxf` program understands the following DXF entities:

Allegro X PCB Editor Entities	DXF Entities
Line, cline	`POLYLINE` ⚠ If writing DXF Revision 14, lines and clines are `LWPOLYLINE` entities.
Arc	`POLYLINE`
Circle	`POLYLINE`
Text	`TEXT`
Voids	`POLYLINE`
Filled rectangle	`SOLID`

Rectangle	POLYLINE
Figure	POLYLINE, CIRCLE (only circular figure)
Via/Pin pad	POLYLINE, CIRCLE(only circular pad) ⚠ If writing DXF Revision 14, vias and pin pads are LWPOLYLINE entities if unfilled and HATCH objects if filled.
Shape	POLYLINE ⚠ If writing DXF Revision 14, shapes are LWPOLYLINE entities if unfilled and HATCH objects if filled.
Padstack, symbol definition	BLOCK
Via, Pin, Symbol instance	INSERT
Drawing extents	$EXTMIN, $EXTMAX
Drawing precision	$LUPREC
Board total thickness	$THICKNESS
CLASS/SUBCLASS mapping	LAYER TABLE ENTRY

Working with a Layer Conversion File

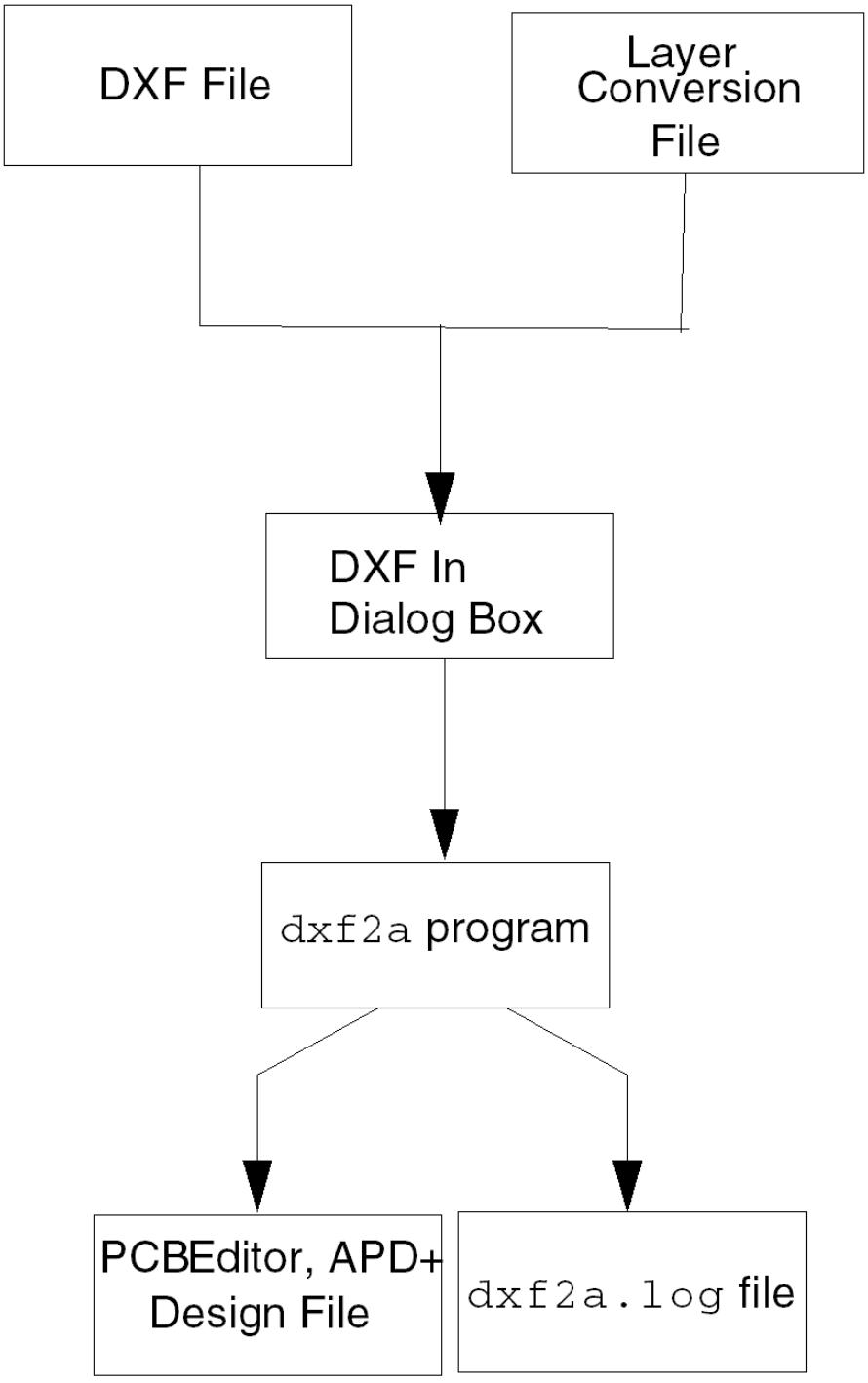
When you import or export DXF data, a layer conversion file maps classes and subclasses to certain DXF layers.

For example, when exporting, if you want only ETCH/CONDUCTOR paths and drafting dimensions to be passed to the DXF output file, you supply only the classes and subclasses for ETCH/CONDUCTOR and dimensions and their corresponding DXF layers.

Importing DXF Data into the Layout Editor

You can import data into a layout design from a DXF file by using *File – Import – DXF* (`dxf in` command) to run the `dxf2a` program. The `dxf2a` program reads the DXF data from the DXF file and creates geometry within a design file. The following figure illustrates the process of DXF to the layout editor.

DXF to Layout Editor



Using the DXF In Dialog Box

When you run the `dxf2a` program using *File – Import – DXF* (`dxf in` command), the DXF In dialog box appears, which lets you specify:

- The original unit of measurement for the DXF design file.

  To avoid round-off issues, do not import with lower accuracy or high-level units than original DXF design.

- The text size blocks as default or user defined.
- The name of the layer conversion file that maps classes and subclasses to specific DXF layers.
- Whether new DXF data is to be added incrementally to a design, that is, without deleting the data already existing in the design.

DXF to Layout Editor Entity Mapping

The `dxf2a` program understands the following DXF entities.

DXF Entities	Corresponding Layout Editor Entities
`LINE`	Line segment for line or cline
`TRACE`	Line segment for line or cline
`ARC`	Arc for line or cline
`CIRCLE`	Circle
`SOLID`	Shape
`TEXT`	Text
`DIMENSION`	Dimension symbol
`ATTDEF, ATTRIB`	Symbol and padstacks definitions only; all others are ignored
`INSERT`	Symbol or via instance
`POLYLINE`	Shape, line, or cline

LWPOLYLINE	Shape, line, or cline
BLOCK	Symbol and padstacks definitions only
$LIMMIN	Used with limmax, extmin, and extmax to determine design extents
$LIMMAX	Used with limmin, extmin, and extmax to determine design extents.
$EXTMIN	Used with limmax, limmin, and extmax to determine design extents
$EXTMAX	Used with limmax, extmin, and limmin to determine design extents
$LUPREC	Design accuracy

Translating Solid Fill

As hatches are not supported in the layout editor, when you want a solid fill to import, the layout editor translates closed polylines to shapes, if the destination subclass supports shapes. Otherwise, the closed polyline becomes a line.

The shape is filled on subclasses that require filled shapes (for example, route keepout). If the subclass does not allow filled shapes, then the shape remains unfilled. Shapes on subclasses that support both filled and unfilled shapes remain unfilled. After translation, you must decide which, if any, should be filled.

This situation occurs most often on the ETCH/CONDUCTOR class when a shape with voids should exist. DXF does not have the concept of voids within a shape; each entity is a closed polyline without any relation to another entity. The *dxf2a* program translates all of these as unfilled shapes. If they were filled, the largest shape would hide all of the other shapes. After translation, you must interactively create the voided shape.

Error Recovery

The import utility may encounter incomplete data, such as no padstack information for a pin/via. In such cases, the utility reports the error in the log file, ignores the entity completely, and continue to the next entity.

When data is erroneous or duplicated, for example, a pin for a via block or multiple padstacks for a pin, the import utility ignores the irrelevant/duplicate entity and proceeds. If the import utility encounters unsupported elements in the DXF file, it generates an error message, and skips the irrelevant data to allow translation to proceed.

Running the `dxf2a` Command in Batch Mode

You can run the `dxf2a` program in batch mode by executing the `dxf2a` command at the console window prompt if you do not want to run the program interactively using the DXF In dialog box that appears when you choose *File – Import – DXF* (`dxf in` command).

Related Topics

- a2dxf
- dxf out
- dxf in
- dxf2a
- DXF Sample File

The GDSII Bi-Directional Manufacturing Interface

The GDSII™ bidirectional manufacturing interface is an IC tool-manufacturing media-storage standard used to:

- Export geometric data from a drawing and convert that data to stream format, which can then be input to GDSII, Construct, or any process that accepts stream input.
- Import geometric data from a GDSII Stream file and create a design file in the layout editor.

The process of importing or exporting stream data involves the following steps:

- Assigning class/subclasses to stream layers by way of a conversion file.
- Exporting data from a physical design and converting it to GDSII stream format using *Manufacture – Stream Out*(GDSII)(`stream out` command).
- Importing stream data to create a design file using *File – Import – Stream*(GDSII) (`load stream` command).

Exporting a Design to GDSII Stream Format

When you export a physical design to GDSII stream format, the layout editor produces a binary file that contains the geometric data filtered onto layers determined by parameters you set in the layer conversion file.

Due to the binary file format, copying the file from one platform to another (for example from Sun® to VAX®) can result in errors because of the differences in floating-point representations. All text is converted to stream format at 0 width. For an area calculation, you must correct text to appropriate height and width. Additionally, rotation differences can occur between text that is mirrored in the layout editor and any system into which the stream format is read.

When you choose *Manufacture – Stream Out* (`stream out` command), it formats internal voids in a the layout editor design into two or more polygons that reproduce the original boundary shape and internal voids, as stream data does not support negative planes nor does it allow internal voids.

All geometric elements are extracted from the layout editor drawing when you use the `stream out` **command** to export stream data. Any lines containing more than 127 line segments are divided into as many paths or boundaries as required to maintain the stream 127 line segment maximum limit.

By default, the layout editor converts all paths, except those representing ETCH/CONDUCTOR, using stream path type 0, which constitutes square-ended pads that end flush with the endpoints. Connect lines are converted using stream path type 1, which constitutes rounded end-points with the center at the digitized points, unless you override the path type by using the `stream_out` batch command options -f, which converts to path type 0, or -s, which converts to path type 2.

⚠ Symbol definitions, negative artwork, and logical information are not transferred.

The following table outlines the mapping between the layout editor elements and stream elements.

Mapping between the Layout Editor and Stream Elements

Layout Editor	Stream
LINE	Path
ARC	Path
CIRCLE	Path
CONNECT LINES	Path
PADS	Path
RECTANGLES	Path
FILLED RECTANGLES	BOUNDARY
SHAPE (including ETCH/CONDUCTOR shapes)	BOUNDARY
TEXT	TEXT
FIGURES	Path

Layer vs. Class/Subclass When Exporting Stream Data

You must assign class/subclasses to stream layers. Assign each class/subclass to a stream layer using a layer conversion file.

Stream Out Layer Conversion Format for Stream Data Export

Column position is unimportant in the format of the stream out layer-to-film conversion table. At least one space must exist between the file name and its corresponding layer. The process converts all film names to uppercase letters before comparing with the layout editor, and appends the `.art` extension.

Using the `stream_out` batch command

You can run `the stream_out` batch command outside the layout editor when you type this batch command at an operating system prompt.

The batch command extracts the film records to create a class/subclass to stream layer filter table. This batch command also uses the stream full-geometry view to extract all geometric information from the layout editor database and converts only those class/subclasses included in the layer filter table.

Arcs and circles are converted to line segments before conversion to stream because stream does not allow arcs and circles.

The command creates one stream structure named STR_1 and includes all the elements in the Allegro X PCB Editor design in this structure.

The `rd_stream` Command

`rd_stream` is a batch commandwhich you can run outside the layout editor. The command outputs data from a stream-format file into standard ASCII text format. You use this ASCII file only to review data. The `rd_stream` command produces an ASCII text file called `stream_file.txt` that you use to review and identify converted data. Because the file is an ASCII representation, it cannot be used as input into any system that reads stream.

ASCII Representation of Stream File

```
Header : GDSII Revision 6.10 : 6 bytes
0006 0002 0262
Bgnlib : 28 bytes
Last Modified Date  3/16/90 14: 8:56
Last Accessed Date  3/16/90 14: 8:56
001c 0102 005a 0003 0010 000e 0008 0038 005a 0003 0010 000e 0008 0038
Libname : small.db : 12 bytes
000c 0206 736d 616c 6c2e 6462
Format : 1 GDSII Filtered Format : 6 bytes
0006 3602 0001
Units :  1000.0000 Database Units per Mil : 20 bytes
0014 0305 3e41 8937 4bc6 a7f0 3a6d 1798 6890 3900
Bgnstr : 28 bytes
Last Modified Date  3/16/90 14: 8:56
Last Accessed Date  3/16/90 14: 8:56
001c 0502 005a 0003 0010 000e 0008 0038 005a 0003 0010 000e 0008 0038
Strname : str_1 : 10 bytes
```

```
000a 0606 7374 725f 3100
Path : 4 bytes 0004 0900Layer :   2 :
6 bytes
0006 0d02 0002
Datatype :   1 : 6 bytes
0006 0e02 0001
Pathtype :   0 : 6 bytes
0006 2102 0000
Width : 567452 : 8 bytes
0008 0f03 0008 8a89cXY : 4 bytes
Coordinates listed in Database Units
X[ 0] : 06200000 , Y[ 0] : 05250000
000c 1003 005e 5e9ac0 0050 501bd0
0004 1003
Endel : 4 bytes
0004 1100
Text : 4 bytes
0004 0c00
Layer :   3 : 6 bytes
0006 0d02 0003
Texttype :   3 : 6 bytes
0006 1602 0003
Width : 16000 : 8 bytes
0008 0f03 0000 3e80
Angle :  0.0000
Degrees : 12 bytes
000c 1c05 0000 0000 0000 0000
XY : 12 bytes
Coordinates listed in Database Units
X[ 0] : 06200000 , Y[ 0] : 05250000
000c 1003 005e 5e9ac0 0050 501bd0
String : U4 : 6 bytes
0006 1906 5534
Endel : 4 bytes
0004 1100
```

Importing Stream Data to Create a Physical Design

You can import geometric data, which includes the stream elements PATH, BOUNDARY, and TEXT, from a GDSII Stream file (`.sf`) to create a physical design.

Importing involves the following:

- Verifying the availability of a stream layer-conversion file by either:
  - Creating a new stream layer conversion file by using a text editor or by using the *S*tream In Edit Layer Mapping dialog box.
  - Editing a layer-conversion file using the *Stream In Edit Layer Mapping* dialog box. You can also use this dialog box to edit stream layer conversion files you created with a text editor.
- Viewing or selecting data on GDSII stream layers for import using the *Layers* tab on the *Stream In View Data* dialog box prior to import.
- Viewing data on GDSII stream structures using the *Structures* tab on the *Stream In View Data* dialog box prior to import. (You cannot selectively choose structures for import.)
- Editing the GDSII Stream Layer Conversion profile to map GDSII stream layers to classes/subclasses using the *Stream In Edit Layer Mapping* dialog box.

Before you import GDSII stream data files into the layout editor, you can selectively view the incoming files either on a layer-by-layer or a structure-by-structure basis. You can select only those layers you wish to import from a particular GDSII stream file, and exclude unwanted layers that the file may contain.

In parallel, you can select layers you ultimately want to import into the layout editor, generate/edit mappings for those layers on the *Stream In Edit Layer Mapping* dialog box, and then import. However, you cannot import data on a structure-by-structure basis.

Sample stream_in.cnv Layer-conversion File

Extra spacing is allowed between fields. A line that begins with a pound sign (#) is a comment line. In this example, the layout editor places all data types (-1) for each stream layer on the manufacturing class/subclass defined in this `stream_in.cnv` layer-conversion file:

```
#sample stream_in.cnv
10 -1 manufacturing 10
20 -1 manufacturing 20
25 -1 manufacturing 25
30 -1 manufacturing 30
35 -1 manufacturing 35
40 -1 manufacturing 40
45 -1 manufacturing 45
50 -1 manufacturing 50
55 -1 manufacturing 55
```

```
60 -1 manufacturing 60
61 -1 manufacturing 61
62 -1 manufacturing 62
```

The layout editor converts the graphical representation of each exported layout editor object into a stream element that is a path, a boundary or a text element. In doing so, the layout editor assigns a data type value to each element that is an arbitrary numeric value.

For example, the layout editor arbitrarily assigns type 4 for wire or cline. The following table shows the data type values for various elements:

Data Type Value (DT)	Element
1	LINE
2	TEXT
3	Endcap Style
4	WIRE/CLINE
5	FIGURE (circle, rectangle, oblong, and so on; primarily for padstack pads)
6	FILLED RECTANGLE
7	RECTANGLE
8	GENERIC SHAPE OUTLINE
9	SHAPE
10	VOID

Other tools may use completely different choices for element type, or may simply set the type of all elements to 0. Other data types found in stream data from other tools may or may not translate. It is possible to separate the data types for a stream layer and place each one on a separate class/subclass in the layout editor. For example, for the stream layer 10 data, instead of entering the following in the `.cnv` file:

```
10 -1 manufacturing 10
```

enter:

```
10 4     manufacturing 10
```

```
10 12    manufacturing 20
10 6     manufacturing 30
```

Import Stream Status and Error Messages

The `stream_in.log` file details the processing status (for example, when processing begins and ends), the library name, and number of entities converted. Review the log file once data has been imported.

Related Topics

- stream out
- load stream
- Creating a Stream Layer Conversion File
- stream_out Batch Command
- rd_stream
- Stream In Edit Layer Mapping Dialog Box
- Viewing and Importing Data on GDSII Stream Layers
- Viewing Data on GDSII Stream Structures
- Mapping or Unmapping GDSII Stream Layers

Intermediate Data Format (IDF)

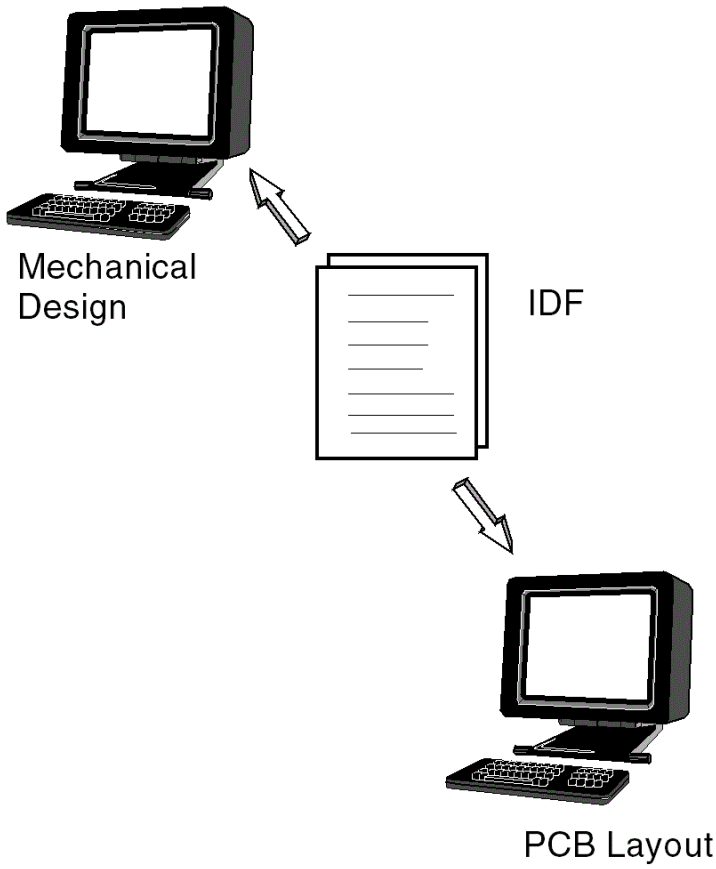
⚠ Although this functionality is available in Allegro X Advanced Package DesignerIDF Board Fil, it is rarely used in these products.

Intermediate Data Format (IDF) is used to translate and exchange data between electrical and mechanical design groups (as seen in the following figure). IDF represents basic design and component geometry information as well as design constraint information such as keepin and keepout regions. The entities supported in IDF are intelligent design entities and are not simply graphical representation of design entities.

For example, a mechanical group designs the board outline and defines critical component placement information. Using IDF, this data is passed to an electrical design group for component placement and routing and then be passed back to the mechanical design group to perform more detailed 3D solid modeling and interference analysis.

Allegro X PCB Editor supports IDF Versions 2.0 and 3.0. See IDF Data Mapping for additional information.

IDF Process



In Allegro X PCB Editor, you can perform these IDF tasks:

- Extract Allegro X PCB Editor data from a design drawing for input to IDF by choosing *File – Export – IDF* (`idf out` command).
- Accept the design outline and component placement information in IDF by choosing *File – Import – IDF* (`idf in` command).

Types of data that can be exchanged include the following:

- Design outline/ Cutout
- Panel outline
- Other outline
- Route outline
- Place outline
- Route keepout
- Via keepout
- Place keepout
- Place region
- Drilled holes
- Notes
- Placement

Filtering Design Objects

To filter objects in the design database for translation and exchange, choose *File – Export – IDF* (`idf_out` command).

A configuration file, `iDfFilterOut.config`, is created in your current working directory. The IDF translator reads this configuration file when you run the command, and excludes the selected objects from the translation. The following figure, an example of the `iDfFilterOut.config` file, shows the objects listed in the file that correspond to the objects you selected for exclusion during translation.

iDfFilterOut.config File Example

```
( filter    Route_Keepout_sym  Route_Keepout_board    Via_Keep-
out_sym  Via_Keepout_board    Plated_Holes  NonPlated_Holes  Vias
Board_Outline  Route_Outline  Package_Keepout  Unplaced_Comp
Placed_Comp  )
```

⚠ When you filter objects for translation and create an `iDfFilterOut.config` configuration file, these settings remain in effect until you reset the filters.

The following table describes the design objects that you can filter from the translator.

Excludable Design Objects

Element Name	Legal Value (Syntax)	Filters Out…
Route Keepout	Route_Keepout_board Route_Keepout_sym	The route keepout shapes owned by board geometry, package symbols, or both
Via Keepout	Via_Keepout_board Via_Keepout_sym	The via keepout shapes owned by board geometry, package symbols, or both
Pins	Plated_Holes Nonplated_Holes	Drill hole information for through-hole pins, either plated or non-plated, or both
Vias	Vias	Drill hole information for vias
Design Outline	Board_Outline	The board outline section of the configuration file
Panel Outline	Panel_Outline	Defines the outline of a circuit board array as it will be either punched or routed during fabrication.
Route Outline	Route_Outline	The route outline section of the configuration file
Package Keepout	Package_Keepout	The package (or place) keepout section of the configuration file
Unplaced Components	Unplaced_Comp	Unplaced components from the placement section of the configuration file
Placed Components	Placed_Comp	Placed components from the placement section of the configuration file

Batch Mode Commands

You can run the `idf out` and `idf_in` commands in batch mode if you do not want to run the commands interactively from within Allegro X PCB Editor.

ECAD/MCAD Ownership Rules

The following table lists the rules that can be directly applied to any IDF-related object that has a unique identifier or unique instance. These include:

- `BOARD_OUTLINE` (only one allowed)
- `PANEL_OUTLINE` (only one allowed)
- `ROUTE_OUTLINE` (only one allowed; layers always set to ALL)
- `PLACE_OUTLINE` (only one allowed; layers always set to ALL; height property does not apply in Allegro X PCB Editor)
- `PLACE_REGION` (identified by name)
- `PLACEMENT` (identified by reference designator)

Rules Applied to IDF-Related Objects

If the IDF Object Owner Is…	And Allegro X PCB Editor…	Then, Allegro X PCB Editor…
`MCAD`	Has no existing object because this is a new design	Imports the object and sets `IDF_OWNER=MCAD`.
`ECAD`	Has no existing object because this is a new design	Imports the object and sets `IDF_OWNER=ECAD`.
`UNOWNED`	Has no existing object because this is a new design	Imports the object and sets `IDF_OWNER=MCAD`.
`MCAD`	Shows the object ownership as `MCAD`	Deletes the existing Allegro X PCB Editor object. It imports the object and sets `IDF_OWNER=MCAD`.

MCAD	Shows the object ownership as ECAD	Rejects the IDF object and reports the rejection in the log file.
MCAD	Shows the object without an IDF_OWNER property	Deletes the Allegro X PCB Editor object. It then imports the object and sets IDF_OWNER=MCAD. If the object has same location in the PCB Editor and MCAD, then this property does not set on the object after import.
ECAD	Shows the object ownership as MCAD	Deletes the existing Allegro X PCB Editor object. It imports the object and sets IDF_OWNER=ECAD.
ECAD	Shows the object ownership as ECAD	Rejects the IDF object and reports the rejection in the log file.
ECAD	Shows the object without an IDF_OWNER property	Deletes the Allegro X PCB Editor object. It then imports the object and sets IDF_OWNER=ECAD.
UNOWNED	Shows the object ownership as MCAD	Deletes the existing Allegro X PCB Editor object. It then imports the object and sets IDF_OWNER=MCAD.
UNOWNED	Shows the object ownership as ECAD	Rejects the IDF object and reports the rejection in the log file.

UNOWNED	Shows the object without an IDF_OWNER property	Deletes the Allegro X PCB Editor object. It then imports the object and sets IDF_OWNER=MCAD.

For IDF-related objects that do not have a unique identifier or unique instance, such as VIA_KEEPOUT, ROUTE_KEEPOUT, and PLACE_KEEPOUT, these additional rules apply:

- Keepouts without an IDF_OWNER property are not deleted.
- IDF keepouts that exactly match Allegro X PCB Editor keepouts are ignored.
- IDF keepouts that do not match existing Allegro X PCB Editor keepouts are imported. This action may create overlapping keepouts.

With the MCAD/ECAD ownership rules, you no longer need to set the idf_nodelete environment variable. If you set this variable, only the PLACEMENT section of the IDF file is processed.

⚠ If IDF_OWNER is set to MCAD when imported into Allegro X PCB Editor, the Allegro X PCB Editor FIXED property is also set to TRUE.

Placement Rules

The following table lists the placement rules for IDF-related objects. If the placement of the incoming part:

- matches the existing placement, then idf_in takes no action. Properties (FIXED or IDF_OWNER) are not added, delete, or modified.
- differs from the existing placement, then idf_in checks for permission to move the symbol (via the IDF_OWNER property), moves the symbol (if permitted), and updates the properties.
- differs, and the movement is not allowed, logs the error.
- has a status of UNPLACED, skips the object.

The placement status of PLACED is considered equivalent to MCAD by idf_in. The layout editor never sets the IDF_OWNER status to PLACED.

Placement Rules Applied to IDF-Related Objects

If the IDF Object Owner Is…	And Allegro X PCB Editor…	Then, Allegro X PCB Editor…
MCAD	Shows the object as UNPLACED	Places and sets `IDF_OWNER=MCAD`.
ECAD	Shows the object as UNPLACED	Places and sets `IDF_OWNER=ECAD`.
UNOWNED	Shows the object as UNPLACED	Places and sets `IDF_OWNER=MCAD`.
MCAD	Shows the object ownership as MCAD	Moves the existing Allegro X PCB Editor object.
MCAD	Shows the object ownership as ECAD	Rejects the IDF object and reports the rejection in the log file.
MCAD	Shows the object without an IDF_OWNER property	Moves the Allegro X PCB Editor object and sets `IDF_OWNER=MCAD`.
ECAD	Shows the object ownership as MCAD	Moves the Allegro X PCB Editor object and sets `IDF_OWNER=ECAD`.
ECAD	Shows the object ownership as ECAD	Rejects the IDF object and reports the rejection in the log file.
ECAD	Shows the object without an IDF_OWNER property	Moves the Allegro X PCB Editor object and sets `IDF_OWNER=ECAD`.
UNOWNED	Shows the object ownership as MCAD	Moves the Allegro X PCB Editor object and sets `IDF_OWNER=MCAD`.
UNOWNED	Shows the object ownership as ECAD	Rejects the IDF object and reports the rejection in the log file.
UNOWNED	Shows the object without an IDF_OWNER property	Moves the Allegro X PCB Editor object and sets `IDF_OWNER=MCAD`

IDF Data Mapping

IDF consists of three files: Board, Library, and Panel (optional). The Board file describes the Printed Wiring Assemblies (PWA) including the board shape, layout restrictions, and component placement. The Library file contains descriptions of the components used by one or more PWAs. Allegro X PCB Editor does not support the optional Panel file.

IDF Board File

For IDF Version 3.0 and the mapping to Allegro X PCB Editor, IDF supports only a closed loop geometry. Geometry is described as a series of coordinate points. IDF requires that the first and last points be identical, thereby closing the loop.

The `.NOTES` section in the Board file allows the exchange of design information and instructions that are not otherwise conveyed by IDF entities. The notes are intended for informational purposes only, not for use in manufacturing drawings. Data in the .NOTES section of IDF imports to the Allegro X PCB Editor subclass `BOARD GEOMETRY/IDF_NOTES`. If there is no subclass, one is created.

Data in the `.OTHER_OUTLINE` section of IDF imports to the user-specified class/subclass. If a formatting error exists, Allegro X PCB Editor places the data in the default subclass, `OTHER_OUTLINE_TOP` or `OTHER_OUTLINE_BOTTOM`, as appropriate.

⚠ To export a shape, rectangle, or filled rectangle into the `.OTHER_OUTLINE` section of IDF, you must attach the `IDF_OTHER_OUTLINE` property.

Although IDF supports multiple keepins, only one keepin is allowed in Allegro X PCB Editor. In instances of multiple keepins, only the first one is imported to the Allegro X PCB Editor database. The rest are discarded with warnings in the log file.

The *Outline height* field in the `PLACEMENT_OUTLINE` section is not written. This implies that there is no height restriction on the outline.

In Allegro X PCB Editor, Package Keepin does not have a height property. When it is exported to the IDF `PLACE_OUTLINE`, the height field is always 0 (zero).

Rooms are allowed on these layers: TOP, BOTTOM, or BOTH.

New fields were added to most IDF sections. Also, new values, added to the *Placement status* field in the PLACEMENT section, specify the system that is the owner: Mechanical (MCAD), Electrical (ECAD), or UNOWNED. Allegro X PCB Editor supports these ownership properties:

- MCAD
  The Mechanical system owns the Outline and should not be modified in the Electrical system.

- ECAD
  The Electrical system owns the Outline and should not be modified in the Mechanical system.
- UNOWNED
  Outline can be modified in either system.

The IDF `PLACEMENT_KEEPOUT` has a single height value that represents the maximum allowable package height. For example, if the value of *Keepout height* equals 300 mils, then packages from 0 through 300 mils in height are allowed, but packages that are greater than 300 mils are excluded.

Allegro X PCB Editor keepouts can have two height values: minimum and maximum. If a keepout has a minimum height value, then the interpretation is:

```
PACKAGE_HEIGHT_MIN = min

PACKAGE_HEIGHT_MAX = INFINITY
```

and the keepout excludes the 3-D space (min, infinity)

Therefore, the appropriate translation from IDF is to set PACKAGE_HEIGHT_MIN = 300 and allow PACKAGE_HEIGHT_MAX to default to INFINITY, thereby excluding the 3-D space (300, infinity).

The `idf_out` utility passes a component's HEIGHT value when both of the following conditions are met:

- No PACKAGE_HEIGHT_MIN and/or PACKAGE_HEIGHT_MAX properties are attached to the place bound shapes defined in the component instance's symbol
- The environment variable `idf_ignore_comp_height` is not set

Mapping IDF to Allegro X PCB Editor

The following table shows the mapping of nine IDF sections to Allegro X PCB Editor. Two other sections, COMPONENTS and DRILLED_HOLES, do not directly map to Allegro X PCB Editor.

IDF to Allegro X PCB Editor Mapping

IDF Section	Allegro X PCB Editor Class/Subclass
BOARD_OUTLINE	BOARD GEOMETRY/DESIGN_OUTLINE
PANEL_OUTLINE	BOARD GEOMETRY/PANEL_OUTLINE
OTHER_OUTLINE	User-specified
ROUTE_OUTLINE	ROUTE KEEPIN
PLACE_OUTLINE	PACKAGE KEEPIN
ROUTE_KEEPOUT	ROUTE KEEPOUT
PLACE_KEEPOUT	PACKAGE KEEPOUT
VIA_KEEPOUT	VIA KEEPOUT
PLACE_REGION	BOARD GEOMETRY/XXX_ROOM
NOTES	BOARD GEOMETRY/IDF_NOTES

Example of IDF Board File

Please refer to `..\algrologic\examples\idf_board.txt`

Other IDF Sections

This section describes the IDF sections that do not directly map to Allegro X PCB Editor.

DRILLED_HOLES

In the IDF DRILLED_HOLES section, only the holes drilled completely through the board are exported from Allegro X PCB Editor to IDF since blind/buried holes are not supported by the IDF format.

When exporting from Allegro X PCB Editor to IDF, holes that are part of a symbol inherit the symbol's `IDF_OWNER` property.

When importing IDF, Allegro X PCB Editor ignores the holes that are associated with a reference designator since they will be created when it places the referenced symbol.

When importing IDF, Allegro X PCB Editor creates a padstack name from the IDF drilled hole parameters using the following formula:

```
<(N)on-plated or (P)lated> + <IDF Hole Type> + <diameter in mils * 10>
```

If a padstack with this name is already defined, then it is used; otherwise a new padstack is defined. An identical symbol name is also created. If a symbol with this name is already defined, then it is used; otherwise a new symbol is defined. This symbol contains a single instance (a pin) of this padstack and places it at the drilled hole's xy location.

PLACEMENT

When Allegro X PCB Editor exports to IDF:

- the IDF *Mounting offset* field is ignored and always defaults to zero.
- the Allegro X PCB Editor symbol name maps to the IDF *Package name* value.
- the Allegro X PCB Editor symbol device maps to the IDF *Part number* value.

When Allegro X PCB Editor imports IDF:

- the IDF component's refdes has priority over the values of *Package name* and *Part number* when placing Allegro X PCB Editor symbols. Any conflicts regarding *Package name* or *Part number* values are logged as warnings.
- if no matching refdes exists, or the IDF component has a value of NOREFDES, then the Allegro X PCB Editor symbol whose name matches the *Package name* is placed. An error is logged if the symbol cannot be found.

IDF Library File

IDF supports only one outline and one height for the component boundary definition. The IDF output produces a worst case outline (overall bounding box) for symbols with multiple placebounds and exports a worst case height for symbols with multiple height definitions, regardless of TOP or BOTTOM assignment. The outline written to the IDF Library file is the union of all geometry that exists on a pair of boundary subclasses. Pre-15.5 release boards may use the `PLACE_BOUND_TOP` and `PLACE_BOUND_BOTTOM` subclasses whereas 15.5 and later boards may use the `DFA_BOUND_TOP` and `DFA_BOUND_BOTTOM` subclasses.

The `DFA_BOUND_TOP` and `DFA_BOUND_BOTTOM` subclasses take precedence over `PLACE_BOUND_TOP` and `PLACE_BOUND_BOTTOM`. But if `idf_out` finds no geometry on the DFA subclasses, then it extracts to the `PLACE_BOUND` subclasses.

To control the place boundary to which to extract to IDF, you can override the component outline subclasses of the Package Geometry class with the environment variables `IDF_PLACE_BOUNDS_TOP` and `IDF_PLACE_BOUNDS_BOTTOM` using *Setup – User Preferences.*

If `idf_out finds` no geometry on the subclasses specified by `IDF_PLACE_BOUNDS_TOP` and `IDF_PLACE_BOUNDS_BOTTOM`, then it uses the `PLACE_BOUND` subclasses. If the `PLACE_BOUND` subclasses do not exist, then `idf_out` uses the `DFA_BOUND_TOP` and `DFA_BOUND_BOTTOM` subclasses.

Example of IDF Library File

```
.HEADER
LIBRARY_FILE         3.0  "allegro 15.1"  2003/05/19.09:50:29        1.00
.END_HEADER
.ELECTRICAL
SMDRES  PRES-10K  THOU      150.00
0      -95.00      -35.00       0.000
0      -95.00       35.00       0.000
0       90.00       35.00       0.000
0       90.00      -35.00       0.000
0      -95.00      -35.00       0.000
.END_ELECTRICAL
.MECHANICAL
OUTLINE  ""  THOU      150.00
.END_MECHANICAL
```

Related Topics

- Exporting Data From a Design Drawing
- IDF_OTHER_OUTLINE
- idf out
- idf_out Batch Command
- idf in
- idf_in Batch Command

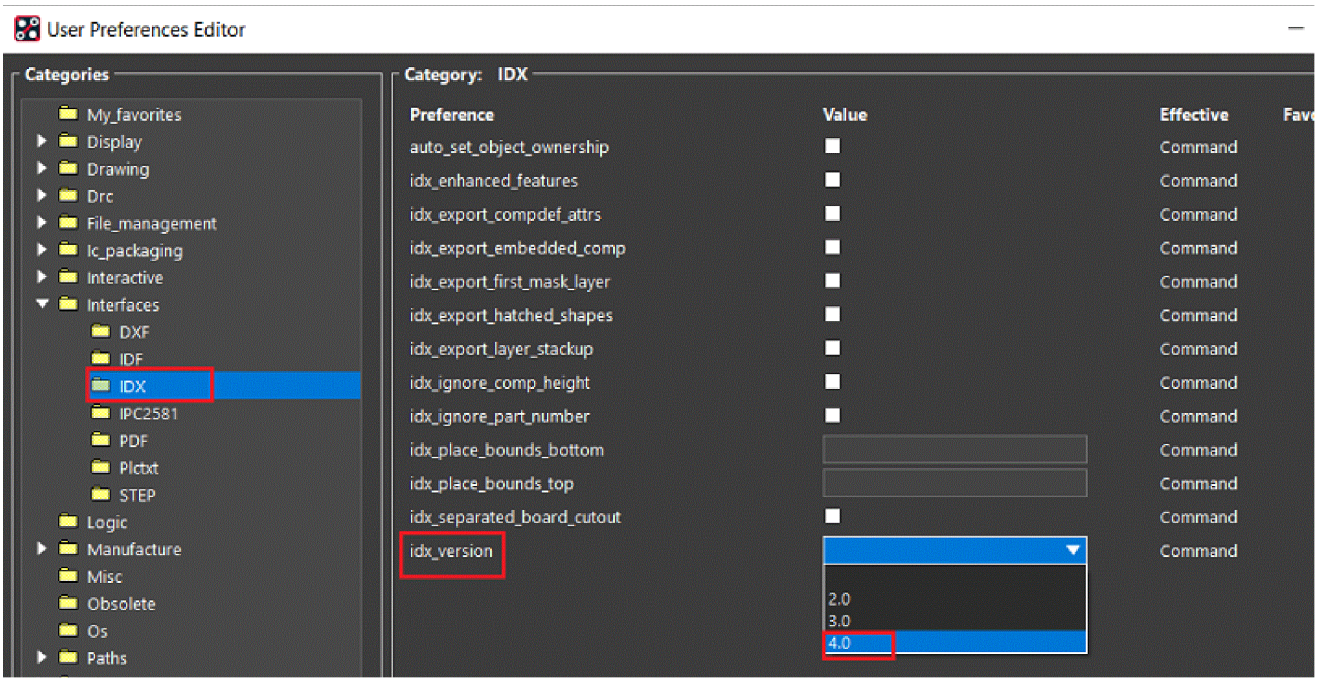
Incremental Data eXchange (IDX) format

ECAD and MCAD designers, traditionally, exchange data using intermediate file formats such as DXF and IDF. The solution is slightly long winded and in every iteration the complete design data needs to be sent.

To enable tighter integration between ECAD and MCAD designers, Cadence now supports the **I**ncremental **D**ata e**X**change (IDX) format.

IDX is an XML-based format that enables you to import and export incremental data into your design. It also facilitates a co-design and collaboration-enabled environment, by providing you an ability to preview the proposed changes before accepting or rejecting them.

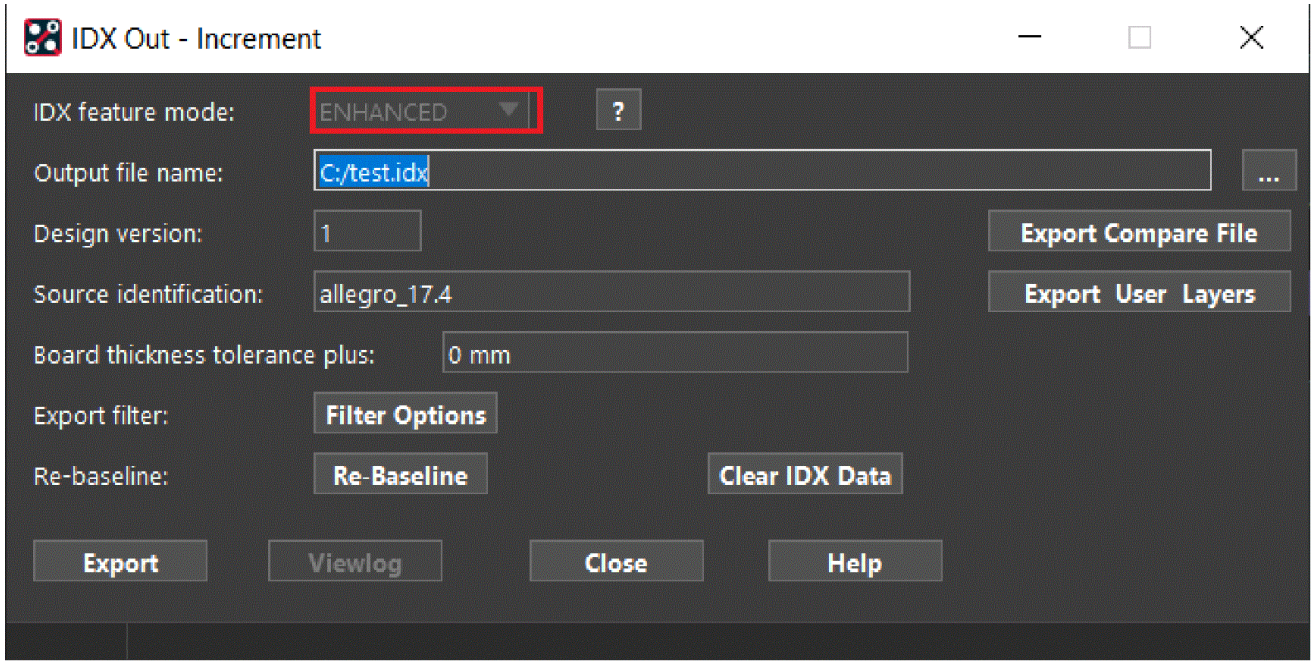
PCB Editor supports IDX `4.0`, `3.0`, and `2.0` versions for creating IDX output. The default version is IDX `3.0`. To enable the latest version, set the value of environment variable *idx_version* to `4.0` in the User Preferences Editor.



Importing IDX data at the start of design process for a design that uses IDX version 4.0 does not require any setup. All import and export actions are by default use 4.0 version.

⚠ If the IDX version is set to the latest for a database, all import and export process remain in version `4.0`. You cannot export the database to lower versions regardless of user preference settings.

By default, the IDX exported the data in STANDARD mode. You can enable the ENHANCED mode by setting the variable *idx_enhanced_features* in the *User Preferences Editor* dialog box. These features are specific to MCAD tool providers. Before enabling these features, verify with MCAD provider which features are supported by their tools.



The supported enhanced feature set includes:

- IDX Properties
- Component Symbol Support
- User Defined Layer and External Copper Layer Support
- IDX Compare Utility

IDX Properties

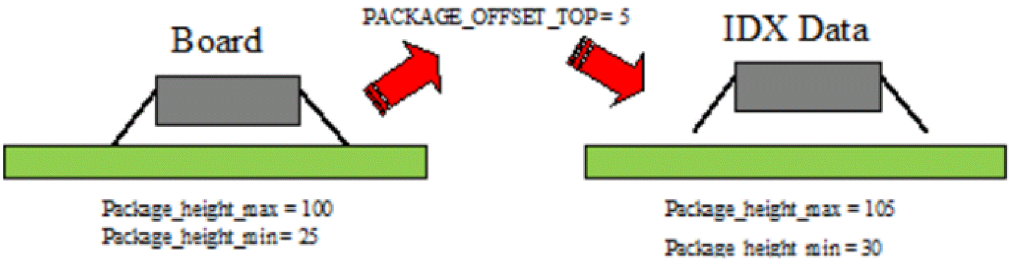
- PACKAGE_OFFSET_BOTTOM and PACKAGE_OFFSET_TOP

In enhanced mode, two board drawing properties are added: `PACKAGE_OFFSET_BOTTOM` and `PACKAGE_OFFSET_TOP`. These properties set the top and bottom offset values for placement and are independent of each other. When exporting IDX data, these properties offsets the component height across the surface of the board that account for paste mask thickness. The value of the property is added to the component height that includes, `PACKAGE_HEIGHT_MIN` and `PACKAGE_HEIGHT_MAX`, height

property or default height values.

> ⚠ If PACKAGE_HEIGHT_MAX property is not specified, the HEIGHT property attached to the component in schematic is used to calculate the package height.

When assigned, the offset is also applied to the symbols when DRC height checks are active.



- IDX_OTHER_OUTLINE

This property allows a bi-directional exchange of the shape(filled and non-filled) if the user-defined shape is not originally defined in the MCAD tool.

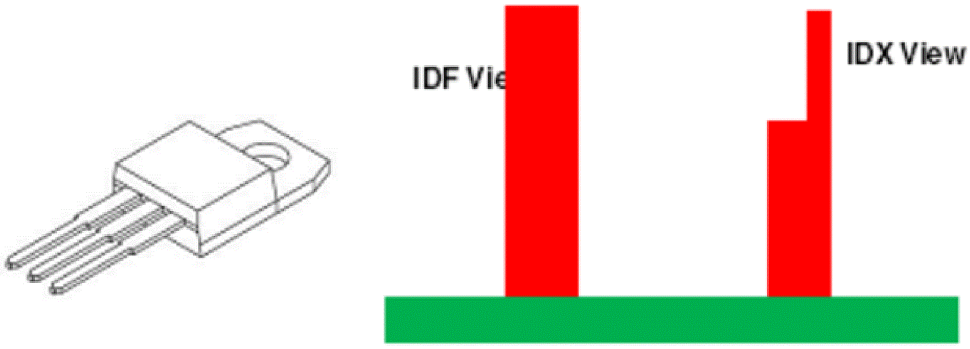
- BOARD_THICKNESS_TOLERANCE_PLUS

This property provides an additional tolerance value to the overall board thickness as an attribute in the IDX data file to the MCAD tool. This value does not impact any data within the board drawing database. You can assign this property through *IDX Out* dialog box.

Component Symbol Support

- Multiple Height Export

The enhanced mode supports multiple `PLACE_BOUND_TOP/BOTTOM` values defined in the package symbol during IDX export. Previously, during export (IDX and IDF) all package boundaries are combined into one unified geometry, and appended to the maximum height value. In enhanced mode, each geometry and associated height values are exported individually within the package symbol definition.



- Pin 1 Identification

In enhanced mode, the `PKG_PIN_ONE` property, assigned to pin 1 of the package symbol, is also exported. The location of pin 1 assists in the placement orientation between the ECAD and MCAD libraries.

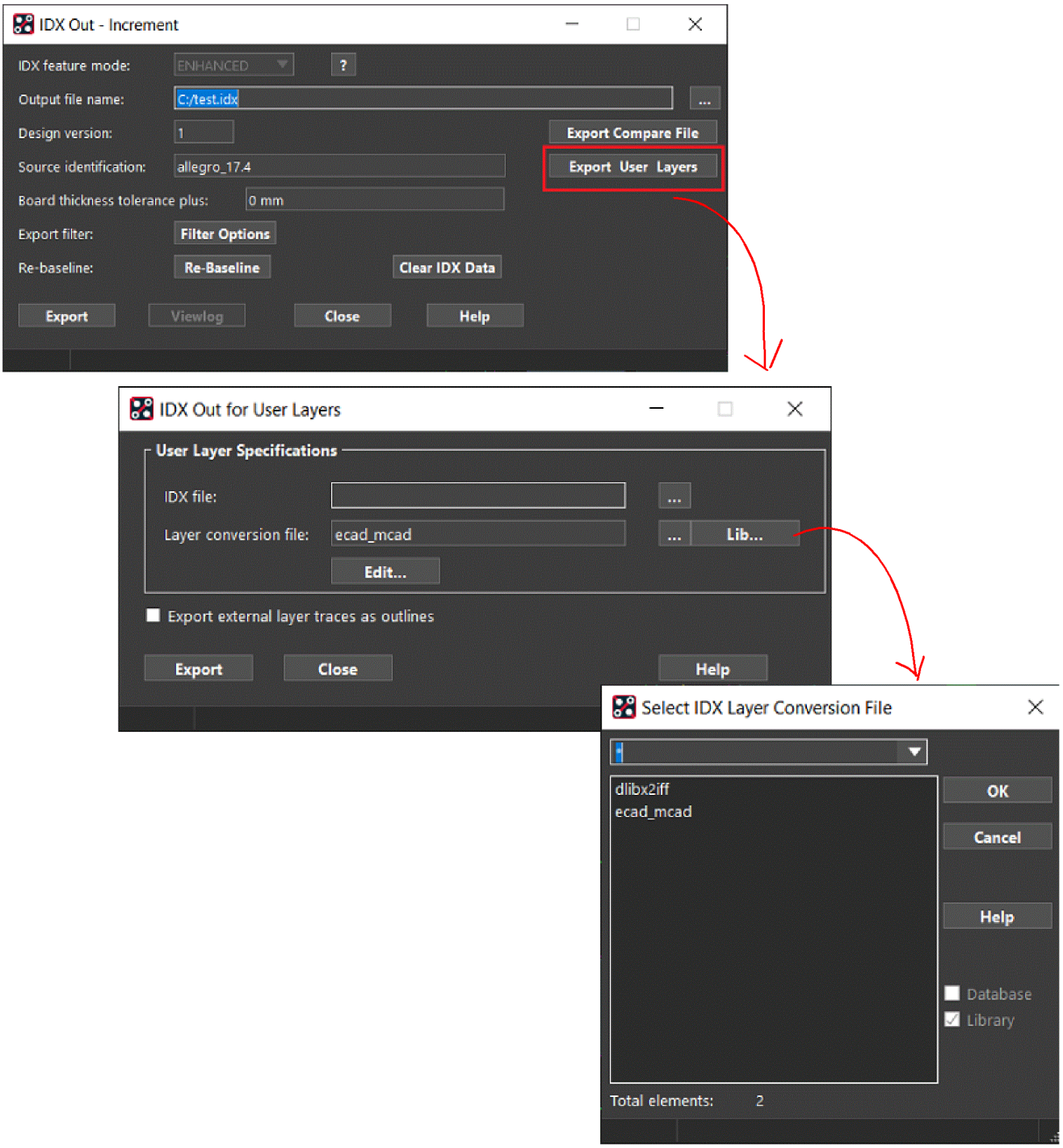
If the `PKG_PIN_ONE` property is not defined, the IDX export utility uses a predefined selection method to determine the primary pin. The order of pin 1 selection is defined in the `pinOneCfg.txt` file located in the installation directory.

User Defined Layer and External Copper Layer Support

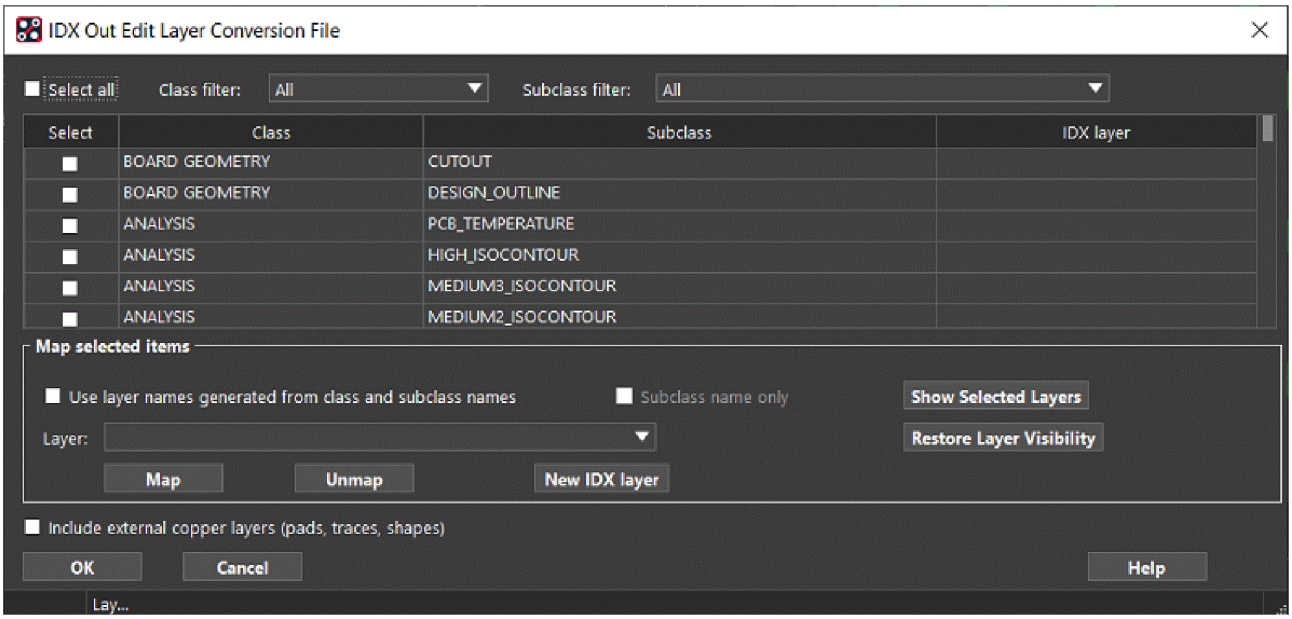
Enhanced features support for IDX includes the ability to import and export user-defined layers and subclasses.

- User-Defined Layer Export

Designs often require export of layers made of different materials and structures, which are not represented by conductor or dielectric such as adhesive layers, plating layers, and so on. The mapping feature available in the *IDX Out* allows you to define Class/Subclasses associated with the MCAD tools layering requirements.



You can associate multiple Class/Subclass with the MCAD layer.



Only those layers that are mapped are exported to a new baseline IDX file. Etch-shapes initially created by the MCAD tool can be imported using the mapping functionality.

- User-Defined Layer Import

Import of user-defined layers is similar to export and uses the same mapping tool. The MCAD layer name defined in the mapping file must match the MCAD layer name in the IDX file. The geometric data contained on that layer is added to the Class/Subclass identified in the mapping file. The layer is mapped with the first mapped instance found in the layer conversion file `ecad_mcad.cnv`. The MCAD layer cannot be mapped to multiple Class/Subclasses.

Example

Sample of a mapping file:

```
#CLASS! SUBCLASS! IDX_LAYER!
#-----------------------------------------------------------------------
BOARD GEOMETRY!
SOLDERMASK_BOTTOM! SM_BOTTOM!
SOLDERMASK_TOP! SM_TOP!
PACKAGE GEOMETRY!
SOLDERMASK_BOTTOM! SM_BOTTOM!
SOLDERMASK_TOP! SM_TOP!
PIN!
SOLDERMASK_BOTTOM! SM_BOTTOM!
SOLDERMASK_TOP! SM_TOP!
```

```
VIA CLASS!
SOLDERMASK_BOTTOM! SM_BOTTOM!
SOLDERMASK_TOP! SM_TOP!
ETCH!
TOP! ETCH_TOP!
BOTTOM! ETCH_BOTTOM!
PIN!
TOP! PAD_TOP!
BOTTOM! PAD_BOTTOM!
VIA CLASS!
TOP! PAD_TOP!
BOTTOM! PAD_BOTTOM!
#END
```

- Hole/Slot Support

Padstacks defined with rounded or rectangle slotted holes are exported as slots in the enhanced mode of the IDX export.

IDX Compare Utility

Exchanging ECAD/MCAD data during the design process often requires many different versions of files that can be mislaid during the collaboration process. Importing an incorrect file might cause ECAD and MCAD design out-of-sync. Based on the IDX collaboration process, the current states of ECAD and MCAD can be compared to verify synchronization of the two databases. There are two options in this verification process depending upon the capabilities of the MCAD tools:

- Allegro IDX MCAD baseline import
- Allegro IDX Export for Compare

Allegro IDX MCAD Compare (Allegro Compare)

The current state of the MCAD tool is compared with the current state of the Allegro X PCB Editor database. The comparison produces a log file and two incremental files if any differences exist. In this process the MCAD tool exports a baseline file that maintains the current IDX identifiers. An incremental file is also created that represents the differences between the two tools. When the differences are imported, the compare utility can be run again for the final validation.

Allegro IDX MCAD Compare (MCAD Compare)

If the ECAD/MCAD databases are to be compared in the mechanical tool, a special baseline file can be created for the current state of the Allegro X PCB Editor drawing that includes all records of past modifications and special object labeling are reset to new values. Using standard baseline data for comparison leads to incorrect results. Allegro X PCB Editor exports a baseline specifically for comparison without impacting any of the changed states or history on all previous collaboration since the last baseline. This exported file is created for use in comparisons only, and should not be used as a standard baseline.

In Allegro, you can either import IDX or IDF.

IDX Import

When you import IDX data to a design and any IDF property is found, a warning is displayed to continue the import with IDF properties.

If you click *Yes*, IDF data is removed from the design and the *IDX In* dialog box is displayed. On importing, IDF owner and `FIXED` properties are cleared if `IDF_OWNER` = `MCAD`.

IDF Import

When you import IDF data to a design that has an IDX baseline, a similar warning is displayed.

On clicking *Yes* the *IDF In* dialog box appears. On importing, the baseline attachment is cleaned and the IDX object IDs, IDX owner properties and, `FIXED` properties are cleared if `IDX_OWNER` = `MECHANICAL`.

The PowerPCB and Pads Layout Interface

These translators import information from Mentor PowerPCB and Pads Layout ASCII database files into Allegro X PCB Editor board databases. It is assumed that the Mentor PowerPCB and Pads Layout databases being translated are placed and routed.

Familiarity with Mentor PowerPCB and Pads Layout is assumed. For additional information, refer to the Mentor PowerPCB and Pads Layout documentation or contact the vendor.

After the translation is complete, load the symbol drawing file (`.dra`) into Symbol Editor for viewing and editing.

The PCAD Translator

The PCAD translator imports information from PCAD, PDIF, and PCB database files into Allegro X PCB Editor board databases. It is assumed that the PCAD databases being translated are completed (placed and routed).

It is further assumed that you are familiar with Altium PCAD and Allegro X PCB Editor. Data types specific to both systems are discussed but not defined or described.

Database Types

The translator reads Altium PCAD version 4, 5, 6, 7 and 8 PDIF database files and writes a Allegro X PCB Editor board database. Due to format differences, other versions of PDIF files cannot be read.

Translation Methodology

Altium PCAD and Allegro X PCB Editor databases have significant differences. Despite these differences, the translator translates all Altium PCAD design data into Allegro X PCB Editor. The Allegro X PCB Editor database created is not structured the same as the PDIF database and most likely does not conform to the data organization recommended by Allegro X PCB Editor. For example, PCAD has no mandatory layer usage guidelines. Therefore, you are free to put the fabrication drawing information on 16 different layers. The Allegro X PCB Editor use model contains guide lines and pre-defined subclasses that can be used to create fabrication drawings. Although the translator allows you to map the data on specific PCAD layers to the pre-defined subclasses, it is not always possible to conform to these standards. The end result is a valid Allegro X PCB Editor database that is capable of producing the same final artwork produced by Altium PCAD.

Some of the information found in the Altium PCAD database is not translated. This information consists mostly of information used by the PCAD design editor and some free flashes.

Altium PCAD manipulates data on specific layers where Allegro X PCB Editor manipulates data on specific class and subclasses. The translator provides a user interface that allows you to direct (map) all Altium PCAD data on a specific layer to a specific Allegro X PCB Editor class and subclass.

Converting a PCAD Database to Allegro X PCB Editor

This section describes the process of converting a Altium PCAD database to a Allegro X PCB Editor board file and the associated translator command line arguments. See *File – Import –CAD Translators – PCAD* (pcad in command) in the *Allegro PCB and Package Physical Layout Command Reference* for procedural information.

Creating a PCAD PDIF Database File

The PCAD job file must be converted to a PCAD PDIF database that can be read by the translator. This file contains all component, component instances and net information found in the PCB database. A PDIF file is self-contained does not require any information from PCAD library files.

See the PCAD documentation for more information.

Importing PCAD Information

After you have created the PDIF database file, you are ready to run the `pcad in` program. See *File – Import – CAD Translators – PCAD* (pcad in command) in the *Allegro PCB and Package Physical Layout Command Reference* for procedural information.

Running the `pcad_in` Command In Batch Mode

The PCAD translator can run in batch mode by specifying all required information on the command line. This removes the need for intervention with the user interface and allows for batch translation of several databases using DOS batch files. `Pcad_in` reads the input file. If all required program arguments are not specified, the *P-CAD to Allegro Translation Options* dialog box displays.

See the pcad in command in the *Allegro PCB and Package Physical Layout Command Reference* for procedural information.

File Generation

When the translator is working, a status dialog box is displayed that shows information about the translation progress. The translation may be canceled by pressing the *Cancel* button on the status form. When the translation is finished the status dialog box is closed. You are notified of errors.

The following files are generated in the output directory.

PCAD_IN.LOG	**The translator log file.**

<name>.brd	The Allegro X PCB Editor board file where name is the name of the PDIF file. For example, VGA.PDF produces VGA.brd.
<name>.TXT	The Allegro X PCB Editor netlist created and used by the translator, where name is the name of the PDIF file.
NETIN.LOG	The log file created by the netin command used by the translator.
<deviceName>.TXT	The Allegro X PCB Editor device files, where deviceName is the name of a device used in the design. One file for each device name is generated.
DEVICES.MAP	An Allegro X PCB Editor device name mapping file used because Altium PCAD allows long device (component) names. You can use this file as a reference to see how names were translated.

Most of these temporary files are generated for the translator to use and remain in the output directory for reference. The Allegro X PCB Editor board file is the key file as it is all that is required to edit the design.

Editing the Database

After the translation is complete, load the board file into Allegro X PCB Editor. Cadence recommends that you perform the following steps to create a design that can be maintained completely within Allegro X PCB Editor. See *File – Import – CAD Translators – PCAD* (pcad in command) in the *Allegro PCB and Package Physical Layout Command Reference* for procedural information.

Translation Notes

The translator scans the PDIF file before the Translation Options dialog box is displayed. All layers defined in the PDIF file are listed in the layer mapping list box. The translator also counts the number of route layers found in the PDIF file. All route layers may not be used, though they were defined in Altium PCAD database as being route layers. In this case you have extra Allegro X PCB Editor subclasses that you can remove. Two extra ETCH/CONDUCTOR subclasses (POWER and GROUND) are created in addition to the TOP, BOTTOM and INTERNALs. The PDIF file does not indicate power planes so it is not known if any are required. If required these layers should be used to map power plane isolation data.

You cannot define new subclass names if the Class field is set to ETCH/CONDUCTOR, PIN or VIA Class.

It is possible to map data from multiple PCAD layers to the same Allegro X PCB Editor subclass

layer. Likewise, it is also possible to map data from a single PCAD layer to all ETCH/CONDUCTOR layers of the targeted class. For example, you may wish a round pad on a padstack to be put on all PIN ETCH/CONDUCTOR layers (TOP, INTERNAL1, INTERNAL2 and BOTTOM.) To do this, select the PIN class and the subclass entry {ALL ETCH}.

Be careful not to map multiple PCAD layers to the same PIN or VIA CLASS subclass. Allegro X PCB Editor allows only one object on a single PIN or VIA CLASS subclass. When more than one data object is mapped to the same padstack subclass, the translator ignores all but the first object. In this case a warning is printed to the log file.

All flashes within a padstack are translated. However, only some flashes placed on the board (outside of a padstack) are translated. All flash shapes except OVAL, THERMAL, TARGET and CUSTOM are translated. If an invalid flash is found at the database level, a warning is printed to the log file indicating the location of the flash. If these flashes are required, you must use the PCAD Allegro X PCB Editor to change the flashes into graphical shapes that can be converted.

The Valor ODB++ Translator

The ODB++ data format creates accurate and reliable manufacturing data for high-quality, Gerber-less manufacturing. The Valor Universal Viewer (VUV) lets you review the OBD++ database; read and view over 20 different photo-tooling file formats; and verify backannotation changes from Valor's manufacturing environment.

Supplied and supported by Valor Computerized Systems© , the ODB++Inside translator lets you output a Allegro X PCB Editor design into a Valor ODB++ database. It contains all CAD/EDA database, assembly, artwork, and manufacturing data.

The Allegro X PCB Editor installation CD does not contain data for the ODB++ Inside package. Because Valor software is not tied to Cadence releases, you must download the latest software from the Valor web site to your local hard drive before you can export Allegro X PCB Editor design data, as described in *File – Export – OBD++ inside* (odb_out command) in the *Allegro PCB and Package Physical Layout Command Reference*.

Once downloaded onto your system, the ODB++ Inside package automatically detects new releases of ODB++. If a new version exists, you can download and install it or continue with the version already installed, as outlined on Valor's download installation instructions web page. For Windows, you download one self-installing `.exe` file(::18 M bytes) to a temporary directory; for Unix, a `.tgz` file of similar size.

The next time you choose *File – Export – OBD++ inside* (odb_out command), the latest version of the ODB++ Inside package you downloaded is activated.

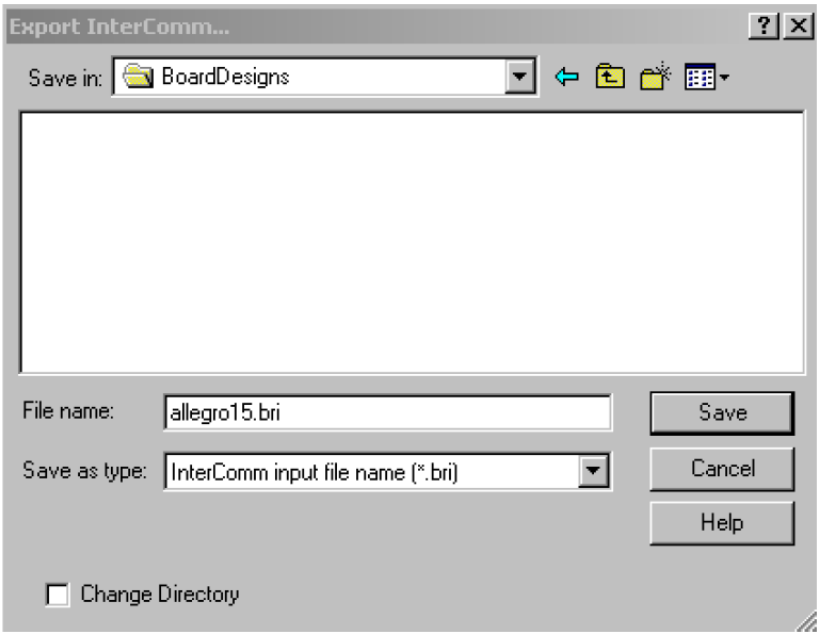
For additional, detailed information on the Valor ODB++ Translator, see the Cadence Board to ODB++ Translator user documentation provided with the Valor software.

InterComm

InterComm is a vendor-independent database viewer and data exchange tool. It facilitates the exchange of schematic and PCB board data among engineering, design, fabrication, test, and assembly departments. Using InterComm's ability to browse through the schematic CAE and PCB CAD databases, query intelligent design data and communicate changes, and comment or red-line data back to the EDA user electronically, the entire product development team can access the corporate Intellectual Property (IP) contained in the logical and physical designs.

With Release 15.0, you can create an extract file from a Allegro X PCB Editor database that serves as input to InterComm. To access InterComm:

1. Choose *File – Export – InterComm* (icm_out command).
   The Export InterComm dialog box appears.
2. Enter the filename and click *Save*.



IPC-D-356

You can export information from the current Allegro X PCB Editor design to an output file that maps directly to the IPC-D-356 format, and supports standard electrical TEST record structure or to the IPC-D-356A format that additionally supports buried and blind via extended records. Allegro X PCB Editor creates an output file with an `.ipc` file extension in your current working directory.

The export functionality applies to all boards developed in version Allegro X PCB Editor 13.0 (formerly Allegro) or later. To export data from earlier versions of Allegro X PCB Editor, you must "uprev" the board to a current version of Allegro X PCB Editor.

During translation, Allegro X PCB Editor creates the *ipc356_out.log* log file in your current working directory. If signal pins are unplaced, Allegro X PCB Editor skips them, and the following message displays at the console window prompt and in the log file:

```
<refdes>-<pin num> on net <net name> is unplaced - SKIPPING.
```

Choose *File – Viewlog* to view the file.

⚠ You can also export IPC 356 data using the `ipc356 out` batch command.

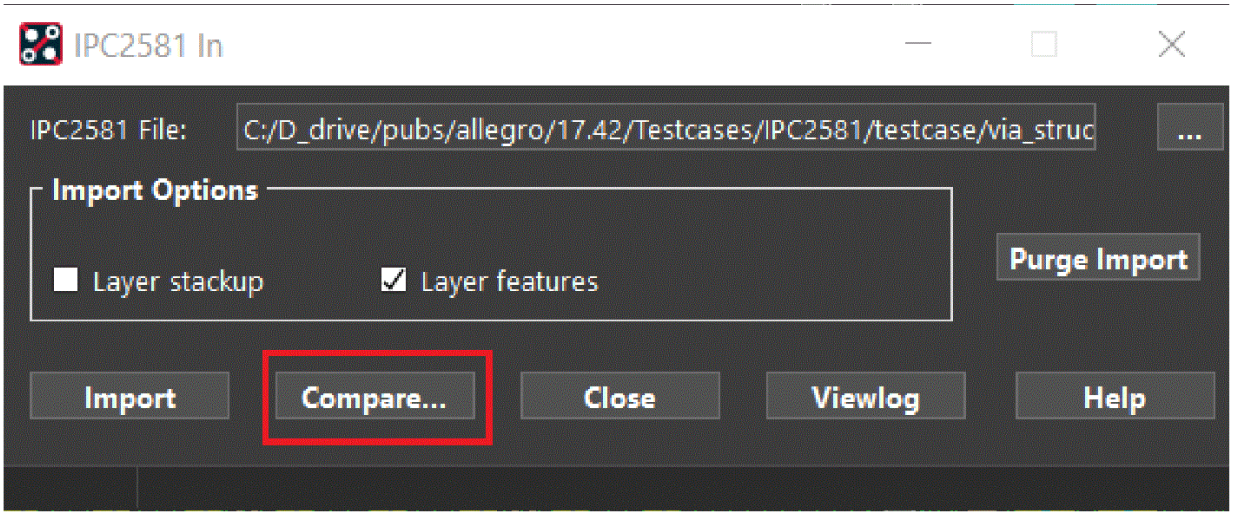
Related Topics

- ipc356 out

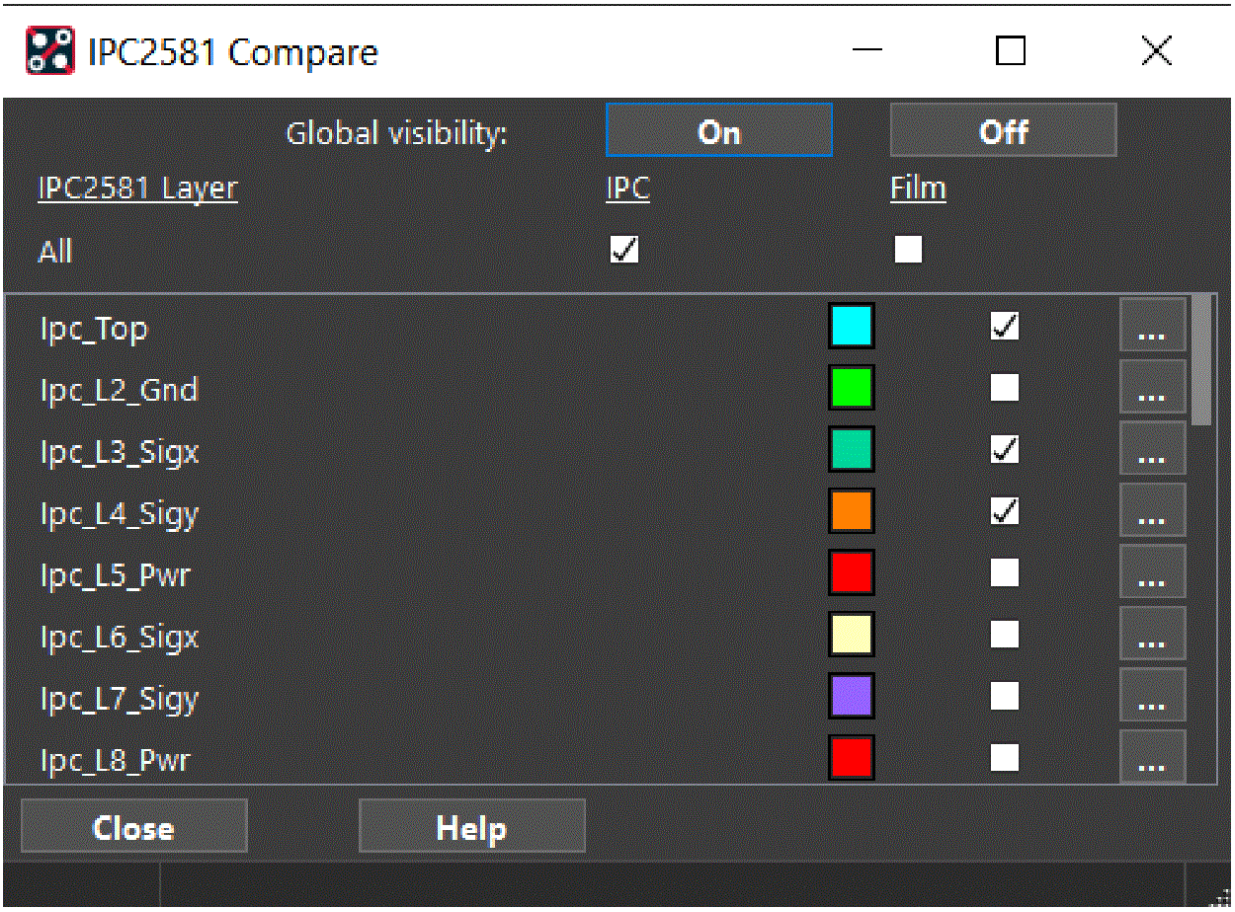
IPC 2581

IPC2581 is a standard data exchange format for using ECAD data in the fabrication and assembly of printed circuit boards. IPC2581 is an XML-based format that contains intelligent data for describing printed board and printed board assembly products with details for tooling, manufacturing, assembly, and inspection requirements. This format is used for transferring information between a printed board designer and a manufacturing or assembly facility.

When import is complete, you can choose *Compare* in the *IPC 2581 In* dialog box to compare the imported layer to the film record subclass layers.



The *IPC2581 Compare* dialog box is displayed.

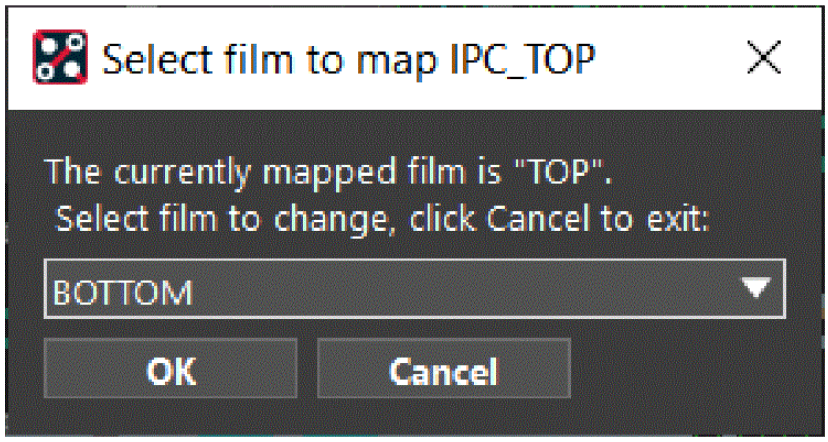


You can visually compare and ensure all artwork data is complete, based on the film record export.

You can also import IPC2581 data into a previous version of a drawing, and visually compare changes from one version of a design to another version.

The *IPC2581 Compare* dialog box lets you:

- Turn ON/OFF the IPC2581 and Film record visibility
- Enable color change pallet with right mouse button
- Browse to select film record to match with the IPC2581 layer



Mentor-to-Allegro PCB Editor Translators

Allegro provides two translators that you can use to convert Mentor data from Mentor Board Station to formats suitable for Allegro:

- The Mentor-to-Allegro X PCB Editor Library Translator lets you convert Mentor libraries (versions C2 and B4) to a format suitable for Allegro X PCB Editor.
- The Mentor-to-Allegro X PCB Editor Board Translator lets you convert Mentor board (versions C2 and B4) to the `.brd` format of Allegro X PCB Editor designs.

Data Mapping

This section describes how the Mentor translators handle different types of data when you translate them to Allegro X PCB Editor.

Layers

Standard Mentor layer specifications are mapped to specific Allegro X PCB Editor subclasses. The translator determines the appropriate class based on the object type being mapped. In the following table, SIGNAL and POWER refer to all signal and/or power layers. The suffix _1 and _2 for appropriate layer specifications indicate top (1) and bottom (2) layers.

Mentor-to-Allegro X PCB Editor Standard Layer Mapping

Mentor Layer Specification	Allegro X PCB Editor Subclasses
`SIGNAL`	Subclasses in ETCH
`POWER`	Subclasses in ETCH
`DRILL`	Drill hole geometry in pads, otherwise treated as non-standard
`PAD {_1, _2]}`	ASSEMBLY_TOP/ASSEMBLY_BOTTOM
`SOLDER_MASK {_1, _2}`	SOLDERMASK_TOP/SOLDERMASK_BOTTOM
`PASTE_MASK {_1, _2}`	PASTEMASK_TOP/PASTEMASK_BOTTOM
`SILKSCREEN {_1, _2}`	SILKSCREEN_TOP/SILKSCREEN_BOTTOM
`PLACE {_1, _2}`	ALL/TOP/BOTTOM
`DRAWING {_1, _2}`	New DRAWING_TOP/DRAWING_BOTTOM subclasses created.

⚠ You can override Mentor DRAWING layers by using a user-defined layer map, described in the following section.

Controlling Layer Mapping

You can exercise greater control of layer mapping through the use of a layer map file, a simple ASCII text file that you can create with any text editor. Layer map files may be necessary in instances where you must override automatic map translations.

The format of the layer map file is a series of entries, one mapping per line, the fields delimited by colons:

```
Mentor Layer Name:Allegro Class:Allegro Subclass
```

The following is an example of a user-defined layer map file containing two entries:

```
ASSEMBLY:PACKAGE GEOMETRY:THERMAL
PLACEMENT_REFERENCE:PACKAGE GEOMETRY:PLACEMENT_REFERENCE
```

Objects

The following table shows how the translator converts standard Mentor objects to Allegro X PCB Editor elements.

Object Translation

Mentor Object	Allegro X PCB Editor Element
`$$create_component`	Component Package
`$$create_generic_part`	Mechanical Package
`$$create_pin`	Pad Stack
`$$create_via`	Via

Geometries

The following table shows how the translator converts standard Mentor geometry types to Allegro X PCB Editor types.

Geometry Translation

Mentor Geometry	Allegro X PCB Editor Geometry
`$$add`	Contents copied into the parent symbols **Note:** Allegro X PCB Editor does not support embedded symbols.
`$$arc`	Arcs
`$$circle`	Circles
`$$path`	Lines/clines (package geometry) or shapes (pad stacks)
`$$polygon`	Shapes
`$$text`	Text

Mentor Attributes

The translator recognizes the following Mentor attributes:

- Pins

aperture_shape_override
terminal_blind_definition
terminal_drill_size
terminal_surface_definition
terminal_thruhole_definition

- Vias

terminal_buried_via_definition
terminal_drill_size
terminal_thruvia_definition

- Components

component_breakout	component_placement_outline
component_default_breakout	component_specific_layer_on
component_default_padstack	component_type
component_edge_connector	drill_definition
component_height	mechanical_parts
component_layout_surface	one-way_region
component_layout_type	required_component_rotation
component_metalization_area	routing_keepout
component_not_in_bom	testpoint_keepout
component_outline_overhang	trace_keepout

component_padstack_override	via_keepout
component_pin_definition	component_insert_center

- Generic parts

board_placement_keepout
breakout_definition_identifier
breakout_via_defintion
drill_definition
drill_definition_unplated
routing_keepout
via_keepout

Miscellaneous Data Types

The following table shows how the translator treats the following Mentor data in Allegro X PCB Editor.

Miscellaneous Data Translation

Mentor	Allegro X PCB Editor
Component placement outlines	PACKAGE GEOMETRY/ASSEMBLY_TOP
`ROUTING_KEEPOUT`	ROUTE KEEPOUT/ALL, TOP or BOTTOM
`TESTPOINT_KEEPOUT`	MANUFACTURING/NO_PROBE (TOP, BOTTOM)
`TRACE_KEEPOUT`	ROUTE KEEPOUT/ALL, TOP or BOTTOM
`VIA_KEEPOUT`	VIA KEEPOUT/ALL, TOP or BOTTOM
Drill definitions	Mechanical padstacks instantiated within packages
Pin escapes	Instantiated as pin escapes where possible
User-defined properties (non-standard $$attribute statements)	Written to the log file

Object Naming

Wherever possible, the translator maintains the names of Mentor objects. However, in instances where special characters and object names do not conform to Allegro X PCB Editor standards, the translator truncates long strings and replaces special characters with underscores. The translator may then modify the object name a second time to insure that it identifies the object uniquely.

Name mapping is accomplished through the use of an ASCII map file, `mentorLibs.map`, that is created in the output directory the first time you run the translator and is modified with each subsequent translation. Because the map file is the basis for naming conventions and is read each time you perform a translation, you must not delete or move the file if you want to perform incremental translations.

Using the Mentor-to Allegro X PCB Editor Library Translator

The Mentor-to-Allegro X PCB Editor Library Translator lets you convert Mentor libraries (versions C2 and B4) to a format that can be used in Allegro X PCB Editor designs. Graphical user interface and batch versions of the translator let you create Allegro X PCB Editor versions of all or part of a library by way of regularly scheduled incremental updates or in a "one shot" complete update of all Mentor-formatted libraries.

You can run the translator in two modes: dynamically, by way of a graphical user interface (the default mode), or through a batch command at the operating system prompt. The translator generates a log file, mentorLibs.log, with each run. The log contains details of the translation process, including:

- Input directories and files
- Output directory
- Translation options used
- List of each object created
- Time and date stamp

The default mode of the translator is a graphical user interface that opens when you type `mbs2lib` at the operating system prompt.

Because the default mode of the translator is through its user interface, you must use the "no GUI" switch to operate in batch mode. See the mbs2lib command in the *Allegro PCB and Package Physical Layout Command Reference* for procedures on running the translator.

Using the Mentor-to Allegro X PCB Editor Board Translator

The Mentor-to-Allegro X PCB Editor Board Translator lets you convert Mentor boards (versions C2 and B4) to a Allegro X PCB Editor .brd file.

You can run the translator through a batch command at the operating system prompt. The translator generates a log file, importMentor.log, with each run. The log contains details of the translation process, including:

- Input directories and files
- Symbols
- Layers
- Padstacks
- Symbol definitions and instances

- Nets, including power and ground nets
- Shapes
- Board geometry
- Connect lines and area fills
- Vias
- Board extents
- Test points
- Constraints

You can see the usage the translator by typing `mbs2brd` at the operating system prompt.

See the mbs2brd command in the *Allegro PCB and Package Physical Layout Command Reference* for procedures on running the translator.

SPD2 and NA2 Format

The SPD2/NA2 translator translates `.spd2` (Cadence Sigrity Unified Package Designer or UPD) and `.na2` (Encore BGA) formats to the `.mcm` format.

You can use the na2 import command (*File – Import – SPD2/NA2)* to open the SPD2/NA2 Import dialog box and then specify the `.na2` or `.spd2` file to be translated.

You can specify to check the syntax of the source file or import only selected information to perform incremental updates using the translator. For incremental updates, cross-section data must match between the `.na2` or `.spd2` file and the `.mcm` files.

⚠ When syntax check of the source file is selected, nothing is imported and the database is not modified.

You can also specify to perform post-processing tasks such as deriving connectivity to detect missing logical associations between overlapping elements, purge unused nets to remove unused logical net names, and update batch DRC to refresh all design rule violations based on the latest constraint information.

Data regarding the following are translated, if available in the file:

- Logical connectivity(netlist)
- Layer stackup
- Padstack definitions
- Constraints (Physical, Spacing, and Electrical)
- Components (Die, BGA, Discrete, and Plating Bar)
- Bond wire and finger placement
- Package substrate routing
- Shapes and planes

The following table lists the mapping of UPD (`.spd2`) layers to APD.

UPD		APD	
CLASS	**SUBCLASS**	**CLASS**	**SUBCLASS**

ARTWORK	BODYOUTLINE	SUBSTRATE GEOMETRY	OUTLINE
ARTWORK	BOTTOMSOLDERMASK	SUBSTRATE GEOMETRY	SOLDERMASK_BOTTOM
ARTWORK	BOTTOMSOLDERPASTE	COMPONENT GEOMETRY	PASTEMAST_BOTTOM
ARTWORK	CONSTRAINTAREAS	CONSTRAINT REGIONS	ALL
ARTWORK	RULER	DRAWING FORMAT	
ARTWORK	TOPSOLDERMASK	SUBSTRATE GEOMETRY	SOLDERMASK_TOP
ARTWORK	TOPSOLDERPASTE	COMPONENT GEOMETRY	PASTEMAST_TOP
DIELECTRIC	BOTAIR	CONDUCTOR	OUTLINE
DIELECTRIC	DIELECTRIC2	CONDUCTOR	DIEELECTRIC2
DIELECTRIC	TOPAIR	DIELCTRIC	
DIELELECTRIC	DIELECTRIC1	CONDUCTOR	DIELECTRIC1
JUMPER	WBOND	DIE/DIESTACK	WIREBOND
SIGNAL	BOTTOM	CONDUCTOR	BOTTOM
SIGNAL	M1	CONDUCTOR	M1

⚠ While importing, names will be changed to be compatible with the Allegro format. For example, all lower-case characters used in the names of net, padstack, component, or layer will be changed to upper-case. If there are conflicts in names, a number is added to the end with an underscore (_), such as `netname_1`.

See the na2 import command in the *Allegro PCB and Package Physical Layout Command Reference* for procedures on running the translator.

Intermediate File Format

The EEsof interface translates Intermediate File Format (IFF) data (including symbols, via structures, and design layers) into the design file. Using the parameters you specify on the *IFF* dialog box, you can automatically generate a route keepout around the subcircuit to keep other connections from introducing noise into the high frequency circuit. Refer to *File – Import – IFF* (iff in command) in the *Allegro PCB and Package Physical Layout Command Reference* for procedural details.

Physical layout affects radio-frequency (RF) circuits that appear as active blocks in digital designs. Adding the RF sub-circuit to a larger design typically requires translating data from one database to another.

You can first design and simulate the high frequency portion of the circuit. Then, after obtaining a special license from Agilent for its EEsof tool, you can generate an Intermediate File Format (IFF) Interface file from the EEsof tool and import the RF sub-circuit into the design. The IFF interface supports `.iff` files generated from either Agilent's ADS or MDS tools. The IFF interface uses the following primary files:

<filename>`.iff`	IFF Input File
`hfsymmap.txt`	Symbol Mapping File
`hflayermap.txt`	Layer Mapping File

To accurately translate data from one database to the other, the IFF interface uses two ASCII mapping files, the layer mapping file and the symbol mapping file.

Layer Mapping File

The Layer Mapping File (`hflayermap.txt`) file maps the EEsof design layers to the physical layers in a design. The layer mapping adheres to the following format:

<EEsof Layer Number> <EEsof Layer Name><Allegro/APD Class/Subclass><Include Flag>

where

<EEsof Layer Number>	Layer number in the EEsof environment
<EEsof Layer Name>	Layer name in the EEsof environment
<Allegro/APD Class/Subclass>	Name of the class/subclass in the Allegro X PCB Editor environment

<Include Flag>	Loads the information contained on this `.iff` layer into the design

Symbol Mapping File

The Symbol Mapping File (`hfsymmap.txt`) symbol mapping file maps the EEsof-developed physical components from the Agilent environment to design symbols, as well as via structures used in EESof to a via padstack in the design.

```
LAYER # IFF LAYER ALLEGRO/APD CLASS/SUBCLASS INCLUDE FLAG
1 default ETCH/TOP T
2 primary ETCH/TOP T
3 sig1 ETCH/TOP T
4 sig2 ETCH/TOP F
5 secondary ETCH/TOP F
6 sig3 ETCH/TOP F
7 sig4 ETCH/TOP F
8 primryres ETCH/TOP F
9 secondres ETCH/TOP F
```

The symbol mapping file adheres to the following format:

`![C|V]!`*`<EEsof Symbol Name>`*`!`*`<Allegro/APD Device Reference>`*`!`*`<X Offset>`*`!`*`<Y Offset>`*

`C`	indicates the record maps a component
`V`	indicates the record maps a via
<EEsof Symbol Name>	name of the EEsof component or via
<Product Device Reference>	name of the product component or via
<X Offset>	x coordinate of spatial placement adjustment (Optional)
<Y Offset>	y coordinate of spatial placement adjustment (Optional)

Sample symbol mapping file

```
!C!RFPPBJTN01!BC848C!37.5!17.0

!C!RFPPRES01!10SC!0.0!15.0

!C!SRLC_RR06GS04_CHC0603!1PFLC!0.0!15.0

!C!SRLC_RR06GS04_CHC0603!0603CAP!0.0!15.0

!C!EEBJT2A_RR06GS04_SOT143!QT-BFP181!29.5!17.0

!V!RR06GS04_VIA!VIA13_G
```

5

Working with Databases

A database is a single file that contains physical and logical information you entered. Databases comprise netlist and device files and include netlist information, component pin information, copies of the symbol files used in the design, and user-defined design and manufacturing rules necessary to create a physical design layout.

Creating a new database requires you to do the following:

- Write a netlist, described below.
- Prepare device files,
- Check the syntax of a netlist using *File – Import – Logic* (`netin` command).
- Load logic data into the layout editor by choosing *File – Import – Logic* (`netin` command).

Updating the logical data in an existing database occurs when the schematic changes, as in an ECO, by running `netin` in either incremental or supersede modes.

Related Topics

- Checking Symbols Automatically
- netin
- Updating a Database.

Writing a Netlist

The layout editor uses a netlist to determine which electrical components the package symbols represent and how signals will be connected. This is the connectivity. The netlist lists the logic functions (NAND gates, resistors, capacitors, connect pins, and so on) or electrical components of the design and their interconnections.

The netlist contains various sections that require the names of the device files for the design:

- The $PACKAGES section identifies the components to be added to the design.
  For each user-created package symbol in the $PACKAGES section of the netlist, provide the name of the device file corresponding to the package symbol and the reference designators for the device type.
- The $FUNCTIONS section identifies the devices with functions.
  You must provide the device file names in the $FUNCTIONS section of the netlist.

The netlist also carries property information for components, functions or nets.

Because the layout editor requires device file information when creating the design database, create device files before using the *Import Logic* dialog box to create the design database.

Create device files using a text editor on your system. You can obtain much of the information for device files from product specification data books for manufacturers.

To generate a device file for a given package symbol use *File – Create Device* menu option in the symbol editor.

You can write a netlist at the gate level (`$FUNCTION`) or at the component level (`$PACKAGES`). You typically describe the circuit at the component level if you are entering the netlist manually, and at the gate level if you have to import the netlist from a third-party front-end tool or if you want to backannotate at the gate level. Netlist sections include:

Netlist Section	Function
$PACKAGES	Specifies the devices in the design. Sometimes it also includes the packages and the components that use them
$FUNCTIONS	Specifies the gates or other parts in the design
$NETS	Specifies the nets in the design and the pins on each net
$PINS	Adds pins to or deletes pins from a net

$END	Terminates the netlist file

The following is a example netlist from Telesis.

```
$PACKAGES
'SSOIC16_0250_01A' ! IXX0884000 ; U27
'RS0402_01A' ! RJX0087400 ! 332 ! '1%' ; R390 R391
$A_PROPERTIES
ROOM 'RMI_XLR' ; R390 R391 U27
$FUNCTIONS
IXX0884000 ! 'U74CB3Q3257DBQ_H2_01' ; F1 F2 F3 F4
RJX0087400 ! RES ! 332 ! '1%' ; F5 F6
$NETS
'+3_3V1' ; U27.16
1N3 ; U27:F4.12
1N4 ; U27:F2.7
'CPU_TO_MCU_SER_N' ; U27:F1.1 U27:F2.1 U27:F3.1 U27:F4.1
GND ; R390:F6.1 R391:F5.1 U27.8
'MCU_RXD0' ; U27:F3.10
'MCU_RXD0_MUX' ; U27:F3.9
'SIO1_MUX_OE_N' ; R390:F6.2 U27:F1.15 U27:F2.15 U27:F3.15 U27:F4.15
'SIO1_MUX_UNUSED' ; R391:F5.2 U27:F2.5 U27:F2.6 U27:F4.13 U27:F4.14
'SIO1_RXD' ; U27:F1.3
'SIO1_TXD' ; U27:F3.11
$A_PROPERTIES
NO_TEST  ; 1N3,1N4
NO_TEST  ; 1N3,1N4
NET_SPACING_TYPE 'HIGH_SPEED_CLKS' ; 'MCU_RXD0_MUX','SIO1_TXD'
IMPEDANCE_RULE 'ALL:ALL:50:10%' ; 'MCU_RXD0_MUX','SIO1_TXD'
$END
```

Use the $PACKAGES section of the netlist to:

- Load component information into a new database so that you can place these components in the layout.
  This information can include the package symbol, device type, component values, and tolerance and reference designators for components.
- Delete component information from a database because these components were removed from the design in an ECO.
- Add component information about components added to the design in an ECO.

- Assign properties to components in a new database or assign properties to components that are the result of an ECO.
- Remove properties from components in response to an ECO.

> ⚠ Component definition properties such as `PART_NUMBER` cannot be added directly in the netlist file and should be added in the device.txt files.

In a netlist that loads design data into a new database, include either a $PACKAGES section or a $FUNCTIONS section. Write a $PACKAGES section in a netlist when you know the footprint packages for components you want to use. A $FUNCTIONS section does not specify packages. A netlist that loads design changes from an ECO does not require a $PACKAGES or a $FUNCTIONS section.

A $PACKAGE section contains:

- Component information, like the device type used by components, that the layout editor adds to a new database.
- $PACKAGES subsection information.

Subsections are for properties and ECOs.

A $PACKAGES section can have the following subsections:

$ADD	Adds components to a database after an ECO
$DELETE	Removes components from a database after an ECO
$A_PROPERTIES	Assigns properties to components
$D_PROPERTIES	Removes a property from components as the result of an ECO

> ⚠ $ADD and $DELETE subsections are used only when you import an incremental netlist. You can import an incremental netlist by un-checking *Supersede all logical data* option in the I*mport Logic* dialog box.

Each line in the $PACKAGES section that is not in a subsection contains fields that add component information to a new database. Some fields are optional; others are required. In the following syntax, the optional items are enclosed in square brackets ([ ]) and ongoing lists are indicated by ellipses (. . .).

```
[packagename] !devicetype [!value[!tolerance]];reference_designator...
```

To start the $PACKAGES section

- Enter `$PACKAGES` on a separate line in the netlist.

To write a line with all fields:

1. Enter a package symbol name.
   This field is optional. It specifies the footprint or package you want the layout editor to use for the components that you specify in subsequent fields in this line.
   If you are writing this netlist manually and know the footprint packages you want to use, enter a package symbol name. The package symbol name you enter here overrides the package symbol name in the device file that is the next field on this line.
   The default name is the name of the package contained in the device file. If none exists, *DIPnn* will be used, where *nn* is the number in the PINCOUNT statement.
   This statement overrides entries in the device file.
   Valid package symbol names have matching `symbol_name.psm` files in the layout editor directory.
2. Enter an exclamation mark.
   This punctuation mark is required even if you did not enter a package symbol name.
3. Enter a device type.
   This field is required. A device type has a matching `device_type.txt` file called a device file.
   The device file specifies device information about the components. This information includes placement class, number of pins, the type of functions in the component, pins that the layout editor can swap with each other, the logical pin use, and the pins for power and ground. The device file also specifies a package, but you can override this specification if you enter a package symbol name at the beginning of this line.
   When it reads and compiles the netlist, the layout editor looks for these device files in the layout editor directory.
4. Enter an exclamation mark.
   This punctuation mark is required only if you are subsequently entering a value field.
5. Enter a value.
   This is an optional field. Some components, like resistors, take an optional value specification. You can use this field to specify, for example, the resistance of the resistors whose reference designators follow on the line.
6. Enter an exclamation mark.
   This punctuation mark is required only if you are subsequently entering a tolerance field.
7. Enter a tolerance.
   This is an optional field. The tolerance specifies the percentage by which the components on

this line can deviate from their specified value. You can enter a blank space for the value field, preceded by an exclamation point, and then enter a tolerance, also preceded by an exclamation point.

8. Enter a semicolon.
   This punctuation mark is required.
9. Enter a list of component reference designators.
   This field is required. Use blank spaces as delimiters between the reference designators in this list. The reference designators in this list use the device file you specified on the line and the package symbol, value, and tolerance if you entered these fields.
   There is no punctuation mark to indicate an end to the list of reference designators. If the list of reference designators makes the line exceed the limit of 78 characters, you can enter a comma between reference designators, then continue the list on the line below.
   If you are writing a netlist to enter design logic from an ECO and you include in a list of reference designators a reference designator that is already in the database, the layout editor omits the reference designator in the list and writes a message in the `netin.log` file.

   ⚠ You specify ranges using brackets.

Sample $PACKAGES section

The following is a sample $PACKAGES section:

```
$PACKAGES
CAPCK05 !'CAPACITOR-1' ; C2 C4 C6 C8
CAPCK05 ! 'CAPACITOR-2' ; C1 C3 C5 C7
CONN10 ! CONNECTOR ; J1 J2 J3
! 74F02 ; N20 N29
DIP14 ! 74F74 ; N05 N08 N11 N14 N17 N23 T11 T14 T17 T20,
 T23 T26 T29 Y05 Y26 Y29
DIP16 ! 74F138 ; N26
DIP16 ! 74F251 ; S05 S08 Y08 Y11 Y14 Y17 Y20 Y23
DIP8 ! LM741 ; H05 H11 H17 H23
RES400 ! 'RESISTOR-1' ; R1 R2 R4 R5 R7 R8 R10 R11 R13
R14, R16 R17 R19 R20 R22 R23
RES400 ! 'RESISTOR-2' ! 33K !; R3 R9 R15 R21
RES400 ! 'RESISTOR-3' !2.2K ! 10%; R6 R12 R18 R24
TO5 ! 2N2222 ; Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8
```

In this example, single quotes enclose some device types. The single quotes are necessary

because these device type names include an invalid character.

Adding Components Required by an ECO

If an ECO adds components to a design, you can use an $ADD subsection in a $PACKAGES section to add information about these components to the database.

The lines in the $ADD subsection use the following syntax:

```
[packagename] !devicetype [!value [!tolerance]]; reference_designator...
```

To write an $ADD subsection:

1. Enter $ADD on a separate line in the $PACKAGES section.
2. Enter lines in the $ADD section using the same method you used to add lines that are not in subsections.

Deleting Components Due to an ECO

If an ECO removes components from a design, you can use a $DELETE subsection of a $PACKAGES section to remove the information about these components from the database.

Each line in the $DELETE subsection specifies components that the layout editor removes from the database. The syntax is

```
reference_designator...
```

To write a $DELETE subsection:

1. Enter $DELETE on a separate line in the $PACKAGES section.
2. On the next line, type a list of component reference designators.
   You can use blank spaces or commas as delimiters between reference designators in these lists.
3. Add additional lines, as needed.

Sample $DELETE subsection

The following is a sample $DELETE subsection:

```
$DELETE
```

```
U1,U2,U3
```

Assigning Properties to Components

You can assign properties to components using an $A_PROPERTIES subsection. See the for a list of default properties. You must add user-defined properties in the layout editor before running `netin` to ensure their inclusion.

The lines in the $A_PROPERTIES subsection specify properties, property values, and the components to which the layout editor assigns these properties. In the following syntax for a line in an $A_PROPERTIES subsection, the optional field is enclosed in square brackets.

propname [*propvalue*] ; *reference_designator* ...

To begin an $A_PROPERTIES subsection:

- Enter $A_PROPERTIES on a separate line in the $PACKAGES section.

To write a `$A_PROPERTIES` line with all fields:

1. Enter a property name.
   This field is required. The layout editor assigns this property to the components you specify subsequently on this line.
2. Enter a property value.
   This field is optional. Enter this field if the property you are assigning requires a value.
3. Enter a semicolon.
   This punctuation mark is required.
4. Enter a list of component reference designators.
   This field is required. It specifies the components to which the layout editor assigns the property.
   Use blank spaces as delimiters between the reference designators in this list. There is no punctuation mark to indicate an end to the list of reference designators. If the list of reference designators makes the line exceed the limit of 78 characters, you can enter a comma between reference designators and continue the list on the line below.

Sample $A_PROPERTIES subsection

The following is a sample $A_PROPERTIES subsection:

```
$A_PROPERTIES
ROOM MEMORY ; U1 U2,U3
HARD_LOCATION TRUE ; U4 U5 U6
```

In this example, the ROOM property is used for quick placement of components that belong together. This property can also be used with a ROOM outline to control the location of components. The `HARD_LOCATION` property prevents the reference designators from being renamed in the layout for critical components, like connectors.

Removing Properties Due to an ECO

You can use a $D_PROPERTIES subsection in a $PACKAGES section to remove a property from a component after an ECO.

The lines in the $D_PROPERTIES subsection specify properties and the components from which they will be removed. The syntax for a line in a $D_PROPERTIES subsection is

propname ; reference_designator ...

To begin a $D_PROPERTIES subsection:

- Enter $D_PROPERTIES on a separate line in the $PACKAGES section.

To write a `$D_PROPERTIES` line:

1. Enter a property name.
   This field is required. It specifies the property that the layout editor removes from the components you subsequently specified on the line.
2. Enter a semicolon.
   This punctuation mark is required.
3. Enter a list of component reference designators.
   This field is required. It specifies the components from which the layout editor removes the property.
   Use blank spaces as delimiters between the reference designators in this list. There is no punctuation mark to indicate an end to the list of reference designators. If the list of reference designators makes the line exceed the limit of 78 characters, you can enter a comma between reference designators and continue the list on the line below.

Sample $D_PROPERTIES section

The following is a sample $D_PROPERTIES section.

```
$D_PROPERTIES
ROOM ; R11 R15 R16 R14 C15
ROOM ; R31 C27 R36 R34 R35,
R38
```

Use a $FUNCTIONS section of a netlist to:

- Load function information into a database.
  This information can include the device type, function type, and the value and tolerance for functions and function designators.
- Add function information to a database about functions added in an ECO.
- Delete function information from a database after these functions were removed in an ECO.
- Assign properties to functions in a new database or assign properties that are the result of an ECO.
- Remove properties from components in response to an ECO.

In a netlist that loads design data into a new database, include either a $FUNCTIONS section or a $PACKAGES section. Include a $FUNCTIONS section to control the selection of the components for functions. A $PACKAGES section does not specify function information. A netlist that loads design changes from an ECO does not require a $PACKAGES or $FUNCTIONS section.

A $FUNCTIONS section contains:

- Function information, like the device type used by specified functions, that the layout editor adds to a new database.
- $FUNCTIONS subsection information.
  Subsections are for properties and ECOs.

A $FUNCTIONS section can have the following subsections:

$ADD	Adds functions to a database after an ECO
$DELETE	Removes functions from a database after an ECO
$A_PROPERTIES	Assigns properties to functions
$D_PROPERTIES	Removes a property from a functions as the result of an ECO

Each line in the $FUNCTION section that is not in a subsection contains fields that add function information to a new database. Just as in the $PACKAGES section, some fields are optional, others are required. In the following syntax, the optional items are enclosed in square brackets ([ ]) and

ongoing lists are indicated by ellipses (...).

```
devicetype [! functiontype [! value [! tolerance ]]] ; function_designator...
```

To begin a $FUNCTIONS section:

- Enter `$FUNCTIONS` on a separate line in the netlist.

To write a line with all fields:

1. Enter a device type.
   This field is required. It specifies the type of device for the component in which the layout editor inserts the function that you specify subsequently in this line.
   A device type has a matching `device_type.txt` file called a device file, that specifies device information about the component.
   When it reads and compiles the netlist, the layout editor looks for these device files in the layout editor directory.
2. Enter an exclamation mark.
   This punctuation mark is required if you enter a function type, value, or tolerance on this line.
3. Enter a function type.
   This is an optional field. If you entered a device type for a homogeneous device (a device that contains only one type of gate), you can omit the function type. If the device type is a heterogeneous device (one that contains more than one type of gate) specify the function type. This prevents the layout editor from swapping two different types of gates with each other.
   If you enter a value or tolerance, enter a function type. If the device type is a homogeneous device, you can enter a blank space for the function type.
4. Enter an exclamation mark.
   This punctuation mark is required if you enter a value or tolerance on this line.
5. Enter a value.
   This is an optional field. Some components, like resistors, take an optional value specification. You can use this field to specify, for example, the resistance of the resistors, whose function designators follow on the line.
6. Enter an exclamation mark.
   This punctuation mark is required only if you are subsequently entering a tolerance field.
7. Enter a tolerance.
   This is an optional field. It specifies the percentage by which the components that contain the functions on this line can deviate from their specified value. You can enter a blank space for

the value, preceded by an exclamation point, and then enter a tolerance, also preceded by an exclamation point.

8. Enter a semicolon.
   This punctuation mark is required.
9. Enter a list of component function designators.
   This field is required. Use blank spaces as delimiters between the function designators in this list.
   A function designator is a name you choose to represent a gate or part in the schematic. The layout editor inserts the functions represented by these function designators into the device whose type you specify on the line.
   The number of function designators is driven by the number of FUNCTION slots (gates) defined in the device file.
   There is no punctuation mark to indicate an end to the list of function designators. If the list of reference designators makes the line exceed the limit of 78 characters, you can enter a comma between reference designators, then continue the list on the line below.

Sample $FUNCTIONS Section

The following is a sample $FUNCTIONS section:

```
$FUNCTIONS
2N2222 ! 2N2222 ; F70 F71 F72 F73 F74 F75 F76 F77
74F02 ! 74F02 ; F114 F115 F116 F117 F118 F119 F120 F121
74F138 ! 74F138 ; F61
74F251 ! 74F251 ; F62 F63 F64 F65 F66 F67 F68 F69
74F74 ! 74F74 ; F82 F83 F84 F85 F86 F87 F88 F89 F90 F91
F92, F93 F94 F95 F96 F97 F98 F99 F100 F101 F102 F103
F104 F105, F106 F107 F108 F109 F110 F111 F112 F113
'CAPACITOR-1' ! 'CAPACITOR-1' ; F4 F5 F6 F7
'CAPACITOR-2' ! 'CAPACITOR-2' ; F0 F1 F2 F3
CONNECTOR ! CONNECTOR ; F32 F33 F34 F35 F36 F37 F38 F39
7423!NOR4 ; F999
7423!NOR4X ; F888
F40, F41 F42 F43 F44 F45 F46 F47 F48 F49 F50 F51 F52 F53
F54 F55, F56 F57 F58 F59 F60
LM741 ! LM741 ; F78 F79 F80 F81
'RESISTOR-1' ! 'RESISTOR-1' ; F16 F17 F18 F19 F20 F21
F22, F23 F24 F25 F26 F27 F28 F29 F30 F31
```

```
'RESISTOR-2' ! 'RESISTOR-2' !22K ; F12 F13 F14 F15
'RESISTOR-3' ! 'RESISTOR-3' !2.2M!10% ; F8 F9 F10 F11
```

In this example some device types and function types are enclosed in single quotes. The single quotes are necessary because these device type and function type names include an invalid character.

Adding Functions Required by an ECO

If an ECO adds functions to a design, you can use an $ADD subsection in a $FUNCTIONS section to add information about these functions to the database.

The lines in the $ADD subsection use the following syntax:

```
devicetype ! functiontype [! value [ tolerance ]]; funcdes
```

To write an $ADD subsection:

1. Enter $ADD on a separate line in the $FUNCTIONS section.
2. Enter lines in the $ADD section using the same method used to add lines that are not in subsections.

Deleting Functions Due to an ECO

If an ECO removes functions from a design, you can use a $DELETE subsection in a $FUNCTIONS section to remove the information about these functions from the database.

Each line in the $DELETE subsection specifies functions that the layout editor removes from the database. The syntax is

function_designator...

To write a $DELETE subsection:

1. Enter `$DELETE` on a separate line in the $FUNCTIONS section.
2. On the next line, enter a list of function designators.
   You can use blank spaces or commas as delimiters between function designators in these lists.
3. Add additional lines, as needed.

Following is a sample $DELETE subsection:

```
$FUNCTIONS
$DELETE
F1,F2,F3
```

You can assign properties to functions using an $A_PROPERTIES subsection. The lines in the $A_PROPERTIES subsection specify properties, property values, and the functions to which the layout editor assigns these properties. In the following syntax for a line in an $A_PROPERTIES subsection, the optional field is enclosed in square brackets.

propname [*propvalue*] ; *function_designator* ...

To begin an $A_PROPERTIES subsection:

- Enter `$A_PROPERTIES` on a separate line in the $FUNCTIONS section.

To write a line with all fields:

1. Enter a property name.
   This field is required. The layout editor assigns this property to the components you specify subsequently on this line.
2. Enter a property value.
   This field is optional. Enter this field if the property you are assigning requires a value.
3. Enter a semicolon.
   This punctuation mark is required.
4. Enter a list of function designators.
   This field is required. It specifies the functions to which the layout editor assigns the property. Use blank spaces as delimiters between the reference designators in this list. There is no punctuation mark to indicate an end to the list of reference designators. If the list of reference designators makes the line exceed the limit of 78 characters, you can enter a comma between reference designators and continue the list on the line below.

The following is a sample $A_PROPERTIES subsection:

```
$A_PROPERTIES
NO_SWAP_GATE TRUE ; F1 F2 F3
NO_SWAP_GATE_EXT TRUE ; F4 F5 F6
```

In this example, the `NO_SWAP_GATE` property prevents swapping of functional gates within the same component. The NO_SWAP_GATE_EXT property allows swapping of functional gates within the same component but prevents with other similar components.

Assigning Reference Designators

You can assign reference designators to components using the `REF_DES_FOR_ASSIGN` property in an $A_PROPERTIES subsection of the $FUNCTIONS section of the netlist.

Specifying the Reference Designator Prefix

Use the `REF_DES_FOR_ASSIGN` property in an $A_PROPERTIES subsection of the $FUNCTIONS section of the netlist.

The following is the syntax for this property in the $A_PROPERTIES subsection:

`REF_DES_FOR_ASSIGN` *`ref_des_prefix ; fuction_designator....`*

The following is a sample line:

```
REF_DES_FOR_ASSIGN U* ; G1 G2 G3 G4
```

In this example, the prefix U is for the reference designator of the component, or components, to which Gate Assign assigns functions G1, G2, G3, and G4.

Specifying the Complete Reference Designator

You can specify the complete reference designator in the netlist instead of just the prefix.

To specify the complete reference designator in the netlist:

- Omit the asterisk in these locations.
  The line that specifies the `REF_DES_FOR_ASSIGN` property in the $A_PROPERTIES subsection of the $FUNCTIONS section.
  The following is a sample line:

  ```
  REF_DES_FOR_ASSIGN U1 ; G1 G2 G3 G4
  ```

  In this example, Gate Assign assigns the reference designator U1 to the component to which it assigns functions G1, G2, G3, and G4.

Removing Properties Due to an ECO

You can use a $D_PROPERTIES subsection in a $FUNCTIONS section to remove a property from a function after an ECO.

The lines in the $D_PROPERTIES subsection specify properties and the components from which the layout editor removes these properties. The syntax for a line in a $D_PROPERTIES subsection is

```
propname ; function_designator ...
```

To begin a $D_PROPERTIES subsection:

- Enter `$D_PROPERTIES` on a separate line in the $FUNCTIONS section.

To write a line:

1. Enter a property name.
   This field is required. It specifies the property that the layout editor removes from the functions you subsequently specify on the line.
2. Enter a semicolon.
   This punctuation mark is required.
3. Enter a list of function designators.
   This field is required. It specifies the functions from which the layout editor removes the property.
   Use blank spaces as delimiters between the function designators in this list. There is no punctuation mark to indicate an end to the list of function designators. If the list of function designators makes the line exceed the limit of 78 characters, you can enter a comma between function designators and continue the list on the line below.

The following is a sample $D_PROPERTIES section.

```
$D_PROPERTIES
NO_SWAP_GATE ; F7 F4,
F1
```

A netlist that loads design data into a new database must include a $NETS section. This information includes the net names and the pins on these nets. A netlist that loads design changes from an ECO does not require a $NETS section. Use the $NETS section of the netlist to:

- Load net information into a new database.
  This net information can include the net names and the pins on the net.
- Specify a routing schedule for a net.
- Remove a routing schedule from a net.
- Delete net information from the database because these nets were removed in an ECO.

- Add net to information about nets added to the design in and ECO.
- Assign properties to nets in a new database of assign properties to nets that are the result of an ECO.
- Remove properties from nets in response to an ECO.
- Specify a reference designator prefix for added components.

A $NETS section contains:

- Net information, such as net names and the pins on a net, that the layout editor adds to a new database
- $NETS subsection information
  Subsections are for routing schedules, properties and ECOs. A $NETS section can have the following subsections:

$SCHEDULE	Schedules a net
$UNSCHEDULE	Removes a schedule from a net
$ADD	Adds nets to a database after an ECO
$DELETE	Removes nets from a database after and ECO
$A_PROPERTIES	Assigns properties to nets
$D_PROPERTIES	Removes a property from nets as the result of an ECO

⚠ $ADD and $DELETE subsections are used only when you import an incremental netlist. You can import an incremental netlist by un-checking *Supersede all logical data* option in the I*mport Logic* dialog box.

Each line in the $NETS section that is not in a subsection contains fields that add net information, net names, and the pins on a net to a new database. You specify the pins on a net with pin designators. There are six valid types of pin designators.

Lines in the $NETS section use one of six formats:

```
[netname] ; reference_designator.pin_number [netname] ;
reference_designator:function_designator.pin_number
reference_designator:function_designator.pin_name:pin_number [netname] ;
function_designator.pin_name [netname] ; function_designator.pin_number
```

To specify the component to which the layout editor assigns a function

- Use a syntax with a pin designator that includes both the component reference designator and the function designator.
  By using these pin designators you assign functions to components. The syntaxes that enable you to assign functions to components are:

```
[netname] ; reference_designator:function_designator.pin_number [netname] ;
reference_designator:function_designator.pin_name [netname] ;
reference_designator:function_designator.pin_name:pin_number
```

The pin designators that include both the reference designator and the function designator are the pin designators most frequently used in third-party software that write a netlist. In pin designators that include both the reference and function designators, the reference designator can be an open series reference designator like U*. An open series reference designator is a prefix for an automatically generated reference designator during gate assignment.

The pin designators that do not list both the reference designator and the function designator are the pin designators most frequently used in netlists that are created manually.

If you are writing a netlist to load design information into a new database and the netlist contains a $FUNCTIONS section and does not include a $PACKAGES section, use a pin designator that contains a function designator. You can use one of the following syntaxes:

```
[netname] ; function_designator.pin_name... [netname] ;
function_designator.pin_number...
```

When you use the pin designator with the pin name and no pin number, the layout editor assigns the pin number during the gate assignment phase of *netin*. (Assigning gates is automatically performed, if needed.) The pin name is all you need to specify a connection to an input or an output of a gate or other type of function in a component.

If you are writing a netlist to load design information into a new database and the netlist contains a $PACKAGES section and does not include a $FUNCTIONS section, use a pin designator that contains a reference designator and a function designator. When writing a netlist by hand you can use the following syntax:

```
[netname] ; reference_designator:function_designator.pin_number
```

An example of such a line is `NET2A ; T*:G2.B  X*:G9.A`

In this example the prefix T is for the reference designator of the component to which *Gate Assign* assigns function G2, and the prefix X is for the reference designator of the component to which Gate Assign assigns function G9.

To specify the complete reference designator

- Omit the asterisk in the pin designator lines of the $NETS section.

The following is a sample line:

```
NET2A ; T1:G2.B  X1:G9.A
```

In this example, the reference designator of the component to which Gate Assign assigns function G2 is T1, and the reference designator of the component to which Gate Assign assigns function G9 is X1.

To begin a $NETS section:

- Enter `$NETS` on a separate line in the netlist.

To write a line in a $NETS section:

1. Enter a net name.
   This is an optional field. If you omit this field, the layout editor assigns a net name that begins with *TN-nnn* where *nnn* is a unique number. Do not enter a net name that begins with *TN-nnn.*
2. Enter a semicolon.
   This punctuation mark is required even if you omit the net name.
3. Enter a list of pin designators.
   The pin designators in this list specify the pins on the net.

Sample $NETS section

The following is a sample $NETS section.

```
$NETS
AGND ; H05.8 H11.8 H17.8 H23.8 J1.1 Q1.1 Q2.1 Q3.1 Q4.1,


 Q5.1 Q6.1 Q7.1 Q8.1 R5.2 R6.1 R11.2 R12.1 R17.2 R18.1 R23.2,
 R24.1
DATA0 ; J2.8 S05.4 S08.4 Y08.4 Y11.4 Y14.4 Y17.4 Y20.4 Y23.4
DATA1 ; C1.2 H05.6 R3.2 S05.3 S08.3 Y08.3 Y11.3 Y14.3 Y17.3,
 Y20.3 Y23.3
DATA2 ; J2.5 S05.2 S08.2 Y08.2 Y11.2 Y14.2 Y17.2 Y20.2 Y23.2
DATA3 ; C3.2 H11.6 R9.2 S05.1 S08.1 Y08.1 Y11.1 Y14.1 Y17.1,
 Y20.1 Y23.1
DATA4 ; J2.7 S05.15 S08.15 Y08.15 Y11.15 Y14.15 Y17.15,
 Y20.15 Y23.15
DATA5 ; C5.2 H17.6 R15.2 S05.14 S08.14 Y08.14 Y11.14 Y14.14,
 Y17.14 Y20.14 Y23.14
```

```
DATA6 ; J2.6 S05.13 S08.13 Y08.13 Y11.13 Y14.13 Y17.13,
 Y20.13 Y23.13
DATA7 ; C7.2 H23.6 R21.2 S05.12 S08.12 Y08.12 Y11.12,
 Y14.12 Y17.12 Y20.12 Y23.12
DCLK ; J2.2 N05.3 N05.11 N08.3 N08.11 N11.3 N11.11 Y05.3,
 Y05.11



ENA1 ; J3.4 N26.2
ENA2 ; J3.8 N26.3
GND ; J3.3 N05.7 N08.7 N11.7 N14.7 N17.7 N20.7 N23.7 N26.4,
N26.5 N26.8 N29.7 S05.7 S05.8 S08.7 S08.8 T11.7 T14.7 T17.7,
 T20.7 T23.7
```

This example uses reference designators and pin numbers in its pin designators. A number of lines exceed 78 characters so they include a comma and continue on the line below.

Scheduling a Net

You can use a $SCHEDULE subsection to specify the routing schedule of a net. When you specify a routing schedule you specify the routing connections between the pins in a net and how the layout editor displays ratsnest lines between these pins.

⚠ Once you schedule a net, the layout editor refers to this schedule as a user schedule.

Each line in the $SCHEDULE subsection specifies the routing schedule of a net. The syntax is:

```
[netname];reference_designator.pin_number...[; reference_designator.pin_number...]
```

To begin a $SCHEDULE subsection:

- Type `$SCHEDULE` on a separate line in the $NETS section.

To write a line with all fields:

1. Enter a net name.
   This field is optional. If you omit the net name, the first pin designator specifies the net on which the layout editor applies the schedule.
2. Enter a semicolon.
   You must use this punctuation mark even if you omit the net name.

3. Enter a list of pin designators.
   The list specifies the order in which the layout editor routes connections between pins and displays ratsnest lines. This list can contain sublists, separated by semicolons, that specify different legs of the schedule. The creation of signal path loops provides the schedules.

The following is an example of a $SCHEDULE subsection. The list of pin designators includes sublists to specify different legs of the schedule.

```
$NETS
$SCHEDULE
 CLK; U1.1 U2.1 U3.1 ; U2.1 U4.1
```

This example includes sublists of pin designators specifying different legs of the schedule.

```
U1.1   U2.1   U3.1
U2.1   U4.1
```

The following diagram shows the two legs of the schedule and the routing connections specified by this schedule.

Scheduling a Net with Tpoints

You can use `$SCHEDULE` to schedule Tpoints (called Ratsnest Ts) in a net. A Tpoint is a point in the physical layout of a net that indicates the signal path splits into multiple paths. You can specify more than one Tpoint in a net, but they must be numbered. The reference in `$SCHEDULE` to `T.1` means the T is now reserved, and you can not use it as a reference designator. The Tpoint `T.1` is unique to the named net. There can be another `T.1` on a different net, as shown in the following example.

The following is an example of the syntax for specifying a Tpoint.

```
$NETS
$SCHEDULE
net1; U1.1 T.1; T.1 U2.2; T.1 U3.3; T.1 U4.4
net2; U2.1 T.1; T.1 U1.2; T.1 U3.2
```

Partially Scheduling a Net

Use $SCHEDULE to partially schedule a net when you want a specific connection order for a portion of a net. Choose pins, multiple series of pins, or Tpoints in a net to specify the order in which to connect to other pins in the net. You can create either subschedule connect points on a pin or Tpoint of a subschedule to designate where to join the remaining net connections. If you do not specify any subschedule connect points, any pin within the subschedule is a legal connection point during net routing. Multiple partial schedules (subschedules) can exist in a net.

The following is an example of a $SCHEDULE subsection for a partial net schedule. The subschedule connect point displays with an asterisk.

```
$NET
$SCHEDULE
NET1; U1.1 *U2.1 U3.1 ; *U2.1 U5.1 ; U8.1 U7.1 ; U8.1 U9.1
```

The following is an illustration of the above $SCHEDULE subsection:

Removing a Schedule from a Net

Use an $UNSCHEDULE subsection to remove routing schedules from nets.

⚠ The only way to delete a Tpoint after its addition to the schedule is to load in a new schedule with $SCHEDULE or choose *Unschedule Pin* from the pop-up menu when you choose *Logic–Net Schedule*.

Each line in the $UNSCHEDULE subsection specifies a net from which the layout editor removes the routing schedule.

You can specify a net in an $UNSCHEDULE subsection in two ways:

- By net name
- By pin designator of a pin in the net

Three different syntaxes exist for lines in an $UNSCHEDULE subsection. One syntax is for a net name, the other two are for a pin designators. These syntaxes are:

```
netname ; reference_designator.pin_number ; function_designator.pin_name
```

To begin an $UNSCHEDULE subsection:

- Type `$UNSCHEDULE` on a separate line in the $NETS section.

To write a line using pin designators:

1. Enter a semicolon.
   This is a required punctuation mark for lines containing pin designators.
2. Enter a list of pin designators.

Sample $UNSCHEDULE Subsection

The following is an example of an $UNSCHEDULE subsection.

```
$UNSCHEDULE
STROBE
; U6.3
CLOCK NOT
```

This example uses both the net name syntax and the pin designator syntax for lines in the $UNSCHEDULE subsection. In this example, a net is named *CLOCK NOT.* This net name includes a blank space. This is not an example of two net names on a line.

Adding Nets Required by an ECO

You can use an $ADD subsection to add nets that result from an ECO.

To begin an $ADD subsection:

- Enter `$ADD` on a separate line in the $NETS section.

The lines in the $ADD subsection specify the information about the nets added to the design by the ECO. These lines can use the same six syntaxes as the lines in the $NETS section that are not in a subsection can use. They are

```
[netname] ; reference_designator.pin_number...
[netname] ; reference_designator:function_designator.pin_number...
[netname] ; reference_designator:function_designator.pin_name...
[netname] ; reference_designator:function_designator.pin_name:pin_number..
[netname] ; function_designator.pin_name...
[netname] ; function_designator.pin_number...
```

The most common syntaxes for $ADD subsections written manually are:

`[netname] ;` *`reference_designator.pin_number...`* `[netname] ;`

function_designator.pin_name... [netname] ; *function_designator.pin_number...*

Sample $ADD Subsection

The following is a sample $ADD subsection:

```
$ADD CLK;U8.10 U1.9 U2.9 U5.9 U6.9
```

Removing Nets After an ECO

You can use a $DELETE subsection to remove nets from a database after an ECO.

There are two ways that you can specify a net for removal in a subsection:

- By net name

–or–

- By pin designator of a pin on the net

There are three types of lines in a $DELETE subsection. One syntax is for a net name; the other two are for a pin designators. They are

netname ; *reference_designator.pin_number* ; *function_designator.pin_name*

To begin a $DELETE subsection:

- Enter `$DELETE` on a separate line in the $NETS section.

To write a line using pin designators:

1. Enter a semicolon.
   This is a required punctuation mark for lines containing pin designators.
2. Enter a list of pin designators.

Sample $DELETE Subsection

The following is a sample $DELETE subsection:

```
$DELETE
STROBE
; U6.3
```

```
CLOCK NOT
```

This example uses both the net name syntax and the pin designator syntax for lines in the $DELETE subsection. In this example, a net is named *CLOCK NOT.* This net name includes a blank space. This is not an example of two net names on a line.

You can assign properties to nets using an $A_PROPERTIES subsection.

The lines in the $A_PROPERTIES subsection specify properties, property values, and the nets to which the layout editor assigns these properties. In the following syntax for a line in an $A_PROPERTIES subsection, the optional field is enclosed in square brackets.

propname [*propvalue*] ; *netname* ...

To begin an $A_PROPERTIES subsection:

- Enter `$A_PROPERTIES` on a separate line in the $NETS section.

To write a line with all fields:

1. Enter a property name.
   This field is required. The layout editor assigns this property to the nets you specify subsequently on this line.
2. Enter a property value.
   This field is optional. Enter this field if the property you are assigning requires a value.
3. Enter a semicolon.
   This punctuation mark is required.
4. Enter a list of net names.
   This field is required. It specifies the nets to which the layout editor assigns the property. Use blank spaces as delimiters between the reference designators in this list. There is no punctuation mark to indicate an end to the list of reference designators. If the list of reference designators makes the line exceed the limit of 78 characters, you can enter a comma between reference designators, then continue the list on the line below.

Sample $A_PROPERTIES Subsection

The following is a sample $A_PROPERTIES subsection:

```
$A_PROPERTIES
ECL TRUE ; PCI_ADDR1 PCI_ADDR2 PCI_ADDR3
MIN_LINE_WIDTH 25MILS ; 1_8V 1_5V 1_25V
MIN_NECK_WIDTH 8MILS ; 1_8V 1_5V 1_25V
```

```
NET_SCHEDULE VERIFY ; DDR3_A1 DDR3_A2 DDR3_A3
RATSNEST_SCHEDULE MIN_DAISY_CHAIN ; DDR3_A1 DDR3_A2 DDR3_A3
VOLTAGE 0V ; GND EGND
```

In this example, the MIN_NECK_WIDTH property is assigned along with the MIN_LINE_WIDTH property to allow flexibility of reducing the cline width when entering and exiting fine-pitch devices. The RATSNEST_SCHEDULE_DAISY_CHAIN property specifies type of ratsnest calculation (ordering) on a given net. The NET_SCHEDULE_VERIFY property generates a DRC if the net is routed outside of its given ratsnest schedule. The VOLTAGE property defines voltage level on a DC net which simplifies the ratsnest display—similar to the NO_RAT property—and is required for power or ground nets for topology extraction.

Removing Properties After an ECO

You can use a $D_PROPERTIES subsection in a $NETS section to remove a property from a net after an ECO.

The lines in the $D_PROPERTIES subsection specify properties and the nets from which the layout editor removes these properties. The syntax for a line in a $D_PROPERTIES subsection is

propname ; netname ...

To begin a $D_PROPERTIES subsection:

- Enter $D_PROPERTIES on a separate line in the $FUNCTIONS section.

To write a line:

1. Enter a property name.
   This field is required. It specifies the property that the layout editor removes from the nets you subsequently specify on the line.
2. Enter a semicolon.
   This punctuation mark is required.
3. Enter a list of net names.
   This field is required. It specifies the nets from which the layout editor removes the property. Use commas as delimiters between the net names in this list. There is no punctuation mark to indicate an end to the list of function designators. If the list of function designators makes the line exceed the limit of 78 characters, you can enter a comma between function designators and continue the list on the line below.

Sample $D_PROPERTIES Section

The following is a sample $D_PROPERTIES section:

```
$D_PROPERTIES
MIN_LINE_WIDTH ; P1J4-A P1J4-C
```

This example removes the `MIN_LINE_WIDTH` property from nets *P1J4-A* and *P1J4-C.*

The contents of a $PINS section are always subsections. Unlike all other sections of a netlist, the $PINS section contains no lines that are not part of a subsection. Use the $PINS section of a netlist to:

- Assign properties to pins.
- Remove properties from pins as the result of an ECO.
- Add pins to a net as the result of an ECO.
- Remove pins from a net as the result of an ECO.

A $PINS section can have the following subsections:

$ADD	Adds pins to a net after an ECO
$DELETE	Removes pins from a net after an ECO
$A_PROPERTIES	Assigns properties to pins
$D_PROPERTIES	Removes properties from pins

⚠ $ADD and $DELETE subsections are used only when you import an incremental netlist. You can import an incremental netlist by un-checking *Supersede all logical data* option in the I*mport Logic* dialog box.

Adding Pins to a Net

Use an $ADD subsection to add pins to a net after an ECO.

The lines in the $ADD subsection specify the pins that the layout editor adds to a net that already exists in a database. There are six types of pin designators so there are six syntaxes for line in the $ADD subsection. They are

```
[netname] ; reference_designator.pin_number...
[netname] ; reference_designator:function_designator.pin_number...
[netname] ; reference_designator:function_designator.pin_name...
[netname] ; reference_designator:function_designator.pin_name:pin_number...
[netname] ; function_designator.pin_name...
[netname] ; function_designator.pin_number...
```

The most commonly used syntaxes in handwritten $ADD subsections are the syntaxes with the shortest pin designators.

In these syntaxes, the net name field is optional. If you omit a net name, the first pin designator in the pin designator list must be a pin that already exists in the database. When you omit the net name, this first pin designator specifies the net on which the layout editor adds all pins designated in the remainder of the pin designator list.

To begin an $ADD subsection:

- Enter `$ADD` on a separate line in the $PINS section.

To write a line with all fields:

1. Enter a net name.
   This field is optional. If you omit it the first pin designator specifies the net.
2. Enter a semicolon.
   This is a required punctuation mark even if you omit the net name.
3. Enter a list of pin designators.

If the $ADD subsection specifies the addition to a net of a pin that already exists on another net in the database, the layout editor writes a message in the `netin.log` file and does not change the pin to the new net.

If the $ADD subsection specifies that a power, ground, or an unconnected pin be included in a net, the layout editor changes the pin to the new net.

Removing Pins from a Net

Use a $DELETE subsection to remove pins from a net after an ECO.

To begin a $DELETE subsection:

- Enter `$DELETE` on a separate line in the $PINS section.

The lines in the $DELETE subsection specify the pins that the layout editor removes from a net. These lines consist of lists of pin designators. The syntax for these lines follows:

```
reference_designator.pin_number...
reference_designator:function_designator.pin_number...
reference_designator:function_designator.pin_name...
reference_designator:function_designator.pin_name:pin_number...
function_designator.pin_name... function_designator.pin_number...
```

Select a syntax based on the design information you know. Most handwritten $DELETE subsections use the shorter pin designators.

Sample $DELETE Subsection

The following is a sample $DELETE subsection:

```
$DELETE
n1.3 n2.1 n2.3
```

In this example, the layout editor removes the pins designated by *n1.3 n2.1* and *n2.3* from their nets.

Assigning Properties to Pins

Use an $A_PROPERTIES subsection to assign properties to pins. See the for a list of default properties.

The lines in the $A_PROPERTIES subsection specify properties, property values, and the pins to which the layout editor assigns these properties. You specify pins with pin designators. There are six types of pin designators but you can only use one type in an $A_PROPERTIES subsection. In the following syntax, the optional field is enclosed in square brackets.

```
propname [propvalue] ; reference_designator.pin_number...
```

To begin an $A_PROPERTIES subsection:

- Enter `$A_PROPERTIES` on a separate line in the $PINS section.

To write a line with all fields:

1. Enter a property name.
   This is a required field. The layout editor assigns this property to the pins you specify subsequently on this line.
2. Enter a property value.
   This field is optional. Enter this field if the property you are assigning requires a value.
3. Enter a semicolon.

This punctuation mark is required.

4. Enter a list of pin designators.
   This field specifies the nets to which the layout editor assigns the property.
   Use blank spaces as delimiters between the pin designators in this list. There is no punctuation mark to indicate an end to the list of pin designators. If the list of pin designators makes the line exceed the limit of 78 characters, you can enter a comma between pin designators and continue the list on the line below.

Sample $A_PROPERTIES Subsection

The following is a sample $A_PROPERTIES subsection:

```
$A_PROPERTIES
PIN_DELAY 1NS ; U1.A1 U1.A2 U1.A3
```

In this example, the `PIN_DELAY` property is assigned to specify the time delay or length of internal package connections for more accurate timing calculations when Propagation Delay, Relative Propagation Delay and Differential Pair Phase Tolerance is used. This value is visible in all the relevant worksheets in the Electrical domain of the Constraint Manager.

⚠ To enable the *Pin Delay,* choose *Setup – Constraints – Modes – Electrical Options.*

Removing Properties

Use a $D_PROPERTIES subsection to remove properties from a pin.

The lines in the $D_PROPERTIES subsection specify properties and the pins from which the layout editor removes these properties. The syntax for a line in a $D_PROPERTIES subsection is:

propname ; reference_designator.pin_number...

To begin a $D_PROPERTIES subsection:

- Enter `$D_PROPERTIES` on a separate line in the $PINS section.

To write a line:

1. Enter a property name.
   This field is required. It specifies the property that the layout editor removes from the pins you subsequently specify on the line.

2. Enter a semicolon.
   This punctuation mark is required.
3. Enter a list of net names.
   This field is required. It specifies the pins from which the layout editor removes the property. Use commas as delimiters between the net names in this list. There is no punctuation mark to indicate an end to the list of function designators. If the list of function designators makes the line exceed the limit of 78 characters, you can enter a comma between function designators and continue the list on the line below.

Sample $D_PROPERTIES Subsection

The following is a sample $D_PROPERTIES subsection:

```
$D_PROPERTIES
NO_PIN_ESCAPE ; U1.3 ,
U1.4 U1.5
```

Related Topics

- Creating a Device File
- *Allegro X Platform Properties Reference*

Creating a Database

After you prepare the netlist and device files, you can use the Import Logic dialog box to check the syntax of a netlist and load the logic data into the layout editor to create a database for a layout. The database contains all the physical and logical data needed for the layout.

See the `netin` command in the *Allegro PCB and Package Physical Layout Command Reference* for procedural information.

Updating a Database

When the schematic changes, as in an ECO, update the logical data in the database by running netin in one of the following modes:

Incremental	Use this mode when the netlist specifies relatively few changes. Such a netlist contains $ADD and $DELETE subsections, and can also contain other types of subsections (such as $A_PROPERTIES). ⚠ The layout editor does not support $SCHEDULE and $UNSCHEDULE subsections.
Supersede	Use this mode when there are extensive changes, or the netlist does not specify the changes. The logical data in this netlist supersedes the logical data in the database: any functions, nets, or components that are not in the netlist are removed from the database.

Using Incremental Mode

To use incremental mode you need a netlist that has changes to the schematic. See the `netin` command in the *Allegro PCB and Package Physical Layout Command Reference* for procedural information. The following is a sample short netlist for adding a resistor in incremental mode:

```
$FUNCTIONS
$ADD
RES1 ! RES1 ! 1000 ; G11
$NETS
$DELETE
NET1
$ADD
NET1A ; G1.B  G11.A
NET1B ; G11.B  G5.A
$END
```

Using Supersede Mode

To use supersede mode, you need a netlist that lists all the logical data in the schematic. You can use a netlist with a $PACKAGES section or a netlist with a $FUNCTIONS section. See the `netin` command in the *Allegro PCB and Package Physical Layout Command Reference* for procedural information.

Using Supersede Mode

Supersede mode works by comparing each net, pin, function, device, component, and symbol in the netlist with those already in the database. When it makes these comparisons:

- If it cannot find an entry in the netlist for a net, pin, function, device, component, or symbol that is in the database, supersede mode deletes that net, function, device, or symbol from the database.
- If it finds a difference between a net, pin, function, device, component, or symbol in the netlist and its counterpart in the database, supersede mode changes the definition of that net, function, device, component, or symbol in the database to match the information in the netlist.

Supersede mode compares nets, pins, functions, devices, components, or symbols. When it finds a difference in properties, either by $A_PROPERTIES or $D_PROPERTIES subsections in the netlist or a new property in the PACKAGEPROP record of a device file specified in the netlist, it adds or deletes these properties from the nets, pins, functions, devices, components, or symbols in the database.

When Comparing Device Information

If the `PACKAGE, CLASS, PINCOUNT, PINORDER, PINUSE, PINSWAP, FUNCTION, POWER, GROUND,` or `NC` record values in the device file specified in the netlist differs from that information for the device in the database, supersede mode:

- Deletes the components that use that device information in the database and adds to the database new components that use the netlist device information.
- Replaces the device file information in the database with the new device file information specified in the netlist and all components for that device use this new device information.

When Comparing Component Information

Supersede mode deletes the existing components in the database unless:

- Their reference designators are in a $PACKAGES section of the netlist
- The netlist assigns functions to their slots by
  - The `REF_DES_FOR_ASSIGN` and `SLOTNAME` properties in an $A_PROPERTIES subsection in the $FUNCTIONS section of the netlist
  - Including function designators, pin names, and pin numbers in the pin designators in the $NETS section

Including the reference designator in the pin designator, without also including the function designator in the pin designator, does not prevent supersede mode from deleting the component.

If the package symbol, device, value, or tolerance in the $PACKAGES section differs from that information in the database, supersede mode deletes these components and adds to the database new components with the new package symbol, device, value, or tolerance.

When Comparing Function Information

Supersede mode deletes:

- All unassigned functions
- All functions that do not have a corresponding function in the netlist
  A netlist specifies a corresponding function with pin designators in the $NETS section that include a reference designator, function designator, pin name, and pin number.
  All functions in the database that have corresponding functions in the netlist remain in the netlist.

When Comparing Pin Information

If a pin is on a net in the database and the netlist specifies a different net, supersede mode adds the pin to the different net. If the netlist specifies no net for that pin, supersede mode removes all associations to nets for that pin.

If you chose *Allow Etch Removal During ECO* and supersede mode changes a pin's net, then supersede mode also rips up the etch connecting to that pin.

6

Comparing Netlists

Design Compare

The Design Compare tool lets you compare physical netlist data from a variety of sources. You can run Design Compare:

- Standalone from an operating system prompt (`design_compare`).
- As a batch comparison tool report from an operating system prompt (`design_compare`).
- From within the layout editor (`design compare`).

With Design Compare, you can load Cadence PCB XML netlist files or import other netlist file types listed below. Then you can display several files for comparison at one time.

You can import netlist data from the following file types:

- Third-Party Netlist File – a netlist imported from a third-party tool using the `netin` command.
- Netlist Report File – a netlist created by running the Netlist report on a design.

  Choose *Tools – Quick Reports – Netlist Report* in PCB Editor, and *Reports – Quick Reports – Netlist Report* in APD to generate this report.
- Netlist with Properties File – a netlist output file that contains pin and net properties for the current design created using the `netout` command.
- Net View Extract File – a netlist created using the `extracta` command.
- Mentor Nets File – a netlist and component list in Mentor format.
- Mentor Neutral File – a Mentor file in ASCII format that provides information about nets, geometry, pins, board locations, drill holes, pads, and testpoints.

To maintain the data in XML format, you can save the netlist files in the Design Compare window.

Accessing Design Compare

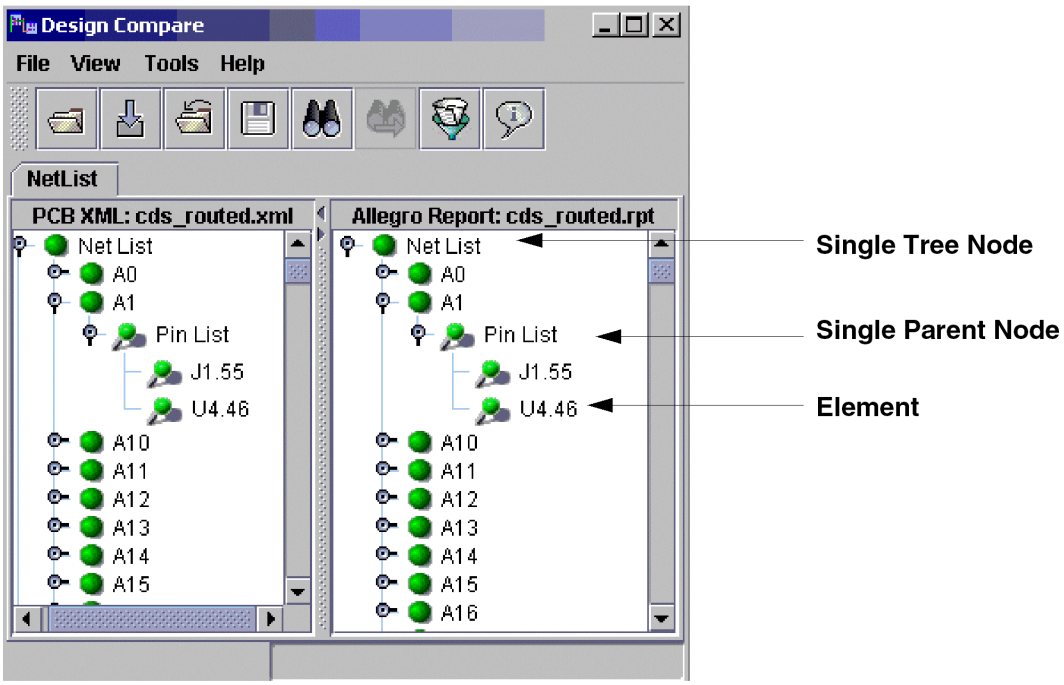
Within the layout editor, choose *Tools – Design Compare* (`design compare` command) to display the Design Compare window.

Design Compare Window

The netlist appears in the Design Compare window as a tree view. Each element is associated with a single tree node. Lists of elements are bundled together into a single parent node.

When you display two netlists, (following figure), they appear in side-by-side split panes. The original netlist appears at the left of the window and becomes the baseline file against which other files are compared; the second, third, and subsequent netlists appear to the right of the window. These netlists are linked so when you select a node in one netlist, the tree scrolls and expands, if necessary, to display the same node in the other netlists.

Design Compare Window



Clicking on a node in the Design Compare tool causes it to expand and display the elements underneath it. Nodes appear as green, red, or yellow.

- Red indicates that there is a difference between the comparable netlist nodes, or that no

comparable node was found.

- Yellow indicates that even though there are no differences between the two nodes, there is a difference between the nodes' descendants (child, grandchild, and so on).
- Green indicates that there are no differences for a node or its descendants.

Navigating the Design Compare Tool

To navigate the Design Compare tool, use the following:

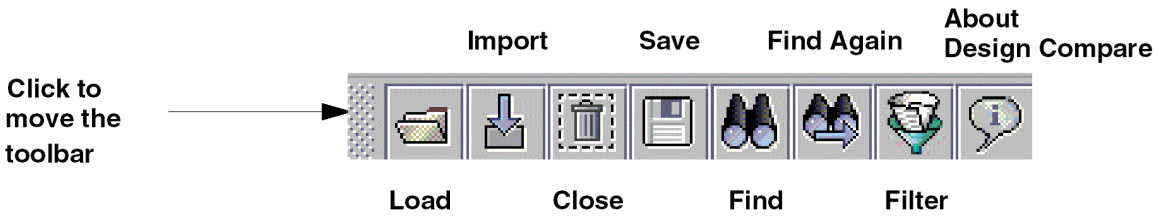
- Design Compare window menus
- Dockable toolbar
- Pop-up menu
- Arrows that appear when there are multiple files in the Design Compare window

Dockable Toolbar

The following figure shows the dockable toolbar. To unlock it, left-click the mouse to the left of the first icon. Then drag the toolbar to the specified area.

If you move the mouse cursor over the bottom right corner of each icon in the toolbar, a description of the icon appears.

Dockable Toolbar



Pop-Up Menu

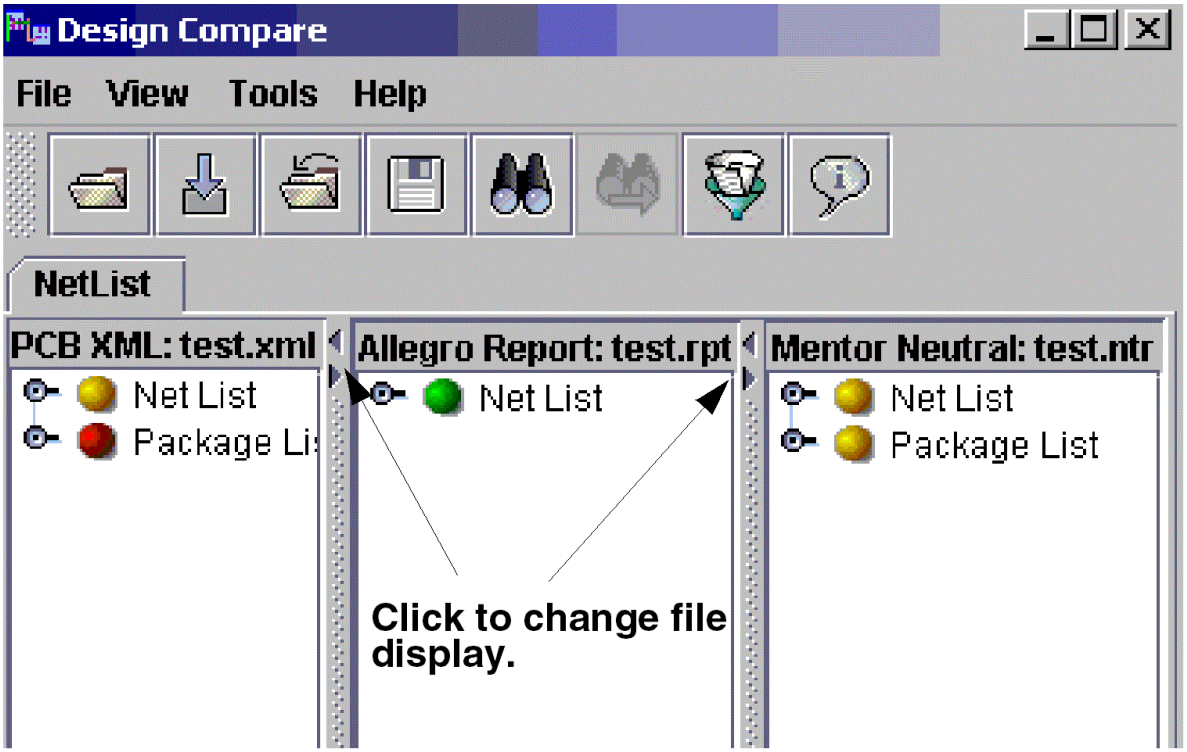
To access the pop-up menu, right-click anywhere in the Design Compare tool.

Design Compare Pop-Up Menu

Arrows for Changing File Display

Using the arrows shown in the following figure, you can manipulate the files shown in the display.

Design Compare Window



Cross-Probing

With the Design Compare tool, once you click on an item in the netlist, the layout editor zooms in on the corresponding component or pin on the displayed board.

Cadence PCB XML DTD

The Data Type Definition (DTD) contains the PCB XML format. This means that those who are familiar with XML protocols can easily read and write PCB XML data.

PCB XML DTD

```
<!------                                                  ====-->
<!------  Cadence PCB Extensible Markup Language DTD      ====-->
<!------            Prototype Version 1.1                 ====-->
<!------                 (cdnpcbml.dtd)                   ====-->
<!------                                                  ====-->
<!------                                                  ====-->
<!------  PCB Collection Elements                         ====-->
<!------                                                  ====-->
<!ELEMENT net_list (net*, package*)>
<!------                                                  ====-->
<!------  Netlist Elements                                ====-->
<!------                                                  ====-->
<!ELEMENT net (id, pin*, property*)>
<!ELEMENT package (id, device*)>
<!ELEMENT device (id, instance*, property*)>
<!ELEMENT instance (id, property*)>
<!ELEMENT pin (id, property*)>
<!------                                                  ====-->
<!------  Generic Elements                                ====-->
<!------                                                  ====-->
<!ELEMENT property (id, value?)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

Related Topics

- design compare
- design_compare
- netin
- netout

Appendix A: DXF Sample File

Following is a sample DXF file.

```
0
SECTION
2
HEADER
9
$ACADVER
1
AC1009
9
$INSBASE
10
0.000000
20
0.000000
9
$LIMMIN
10
-5.000000
20
6.000000
9
$LIMMAX
10
12.000000
```

```
12.000000
20
6.000000
9
$EXTMIN
10
-5.000000
20
-5.000000
9
$EXTMAX
10
12.000000
20
6.000000
9
$ATTMODE
70
0
9
$LUNITS
70
2
9
$LUPREC
70
3
9
```

```
$THICKNESS
40
0.000280
0
ENDSEC
0
SECTION
2
TABLES
0
TABLE
2
LTYPE
70
0
0
LTYPE
2
CONTINUOUS
3
Solid line
70
0
72
65
73
0
```

```
40
0.000000
0
ENDTAB
0
TABLE
2
LAYER
70
23
0
LAYER
2
BOARD_GEOMETRY_OUTLINE
70
0
62
0
6
CONTINUOUS
0
LAYER
2
ETCH_TOP
70
0
62
```

```
1
6
CONTINUOUS
0
LAYER
2
ETCH_BOTTOM
70
0
62
2
6
CONTINUOUS
0
LAYER
2
PACKAGE GEOMETRY_MODULES
70
0
62
3
6
CONTINUOUS
0
LAYER
2
PACKAGE GEOMETRY_DISPLAY_BOTTOM
70
```

```
70
0
62
4
6
CONTINUOUS
0
LAYER
2
PACKAGE GEOMETRY_DISPLAY_TOP
70
0
62
5
6
CONTINUOUS
0
LAYER
2
PACKAGE_GEOMET_SOLDERMASK_BOTT
70
0
62
6
6
CONTINUOUS
0
LAYER
```

```
2
PACKAGE GEOMETRY_SOLDERMASK_TOP
70
0
62
7
6
CONTINUOUS
0
LAYER
2
PACKAGE GEOMETRY_BODY_CENTER
70
0
62
8
6
CONTINUOUS
0
LAYER
2
PACKAGE_GEOMET_SILKSCREEN_BOTT
70
0
62
9
6
```

```
CONTINUOUS
0
LAYER
2
PACKAGE GEOMETRY_SILKSCREEN_TOP
70
0
62
10
6
CONTINUOUS
0
LAYER
2
PACKAGE GEOMETRY_PAD_STACK_NAME
70
0
62
11
6
CONTINUOUS
0
LAYER
2
PACKAGE GEOMETRY_PIN_NUMBER
70
0
```

```
62
12
6
CONTINUOUS
0
LAYER
2
PACKAGE_GEOMETR_ASSEMBLY_BOTTO
70
0
62
13
6
CONTINUOUS
0
LAYER
2
PACKAGE GEOMETRY_ASSEMBLY_TOP
70
0
62
14
6
CONTINUOUS
0
LAYER
2
```

```
REF_DES_DISPLAY_BOTTOM
70
0
62
15
6
CONTINUOUS
0
LAYER
2
REF_DES_DISPLAY_TOP
70
0
62
16
6
CONTINUOUS
0
LAYER
2
REF_DES_SILKSCREEN_BOTTOM
70
0
62
17
6
CONTINUOUS
0
```

```
LAYER
2
REF_DES_SILKSCREEN_TOP
70
0
62
18
6
CONTINUOUS
0
LAYER
2
REF_DES_ASSEMBLY_BOTTOM
70
0
62
19
6
CONTINUOUS
0
LAYER
2
REF_DES_ASSEMBLY_TOP
70
0
62
20
```

```
6
CONTINUOUS
0
LAYER
2
BOUNDARY_TOP
70
0
62
21
6
CONTINUOUS
0
LAYER
2
BOUNDARY_BOTTOM
70
0
62
22
6
CONTINUOUS
0
ENDTAB
0
TABLE
2
```

```
STYLE
70
0
0
STYLE
2
STANDARD
70
0
40
0.000000
41
1.000000
50
0.000000
71
0
42
0.200000
3
txt
0
ENDTAB
0
ENDSEC
0
SECTION
```

```
2
BLOCKS
0
BLOCK
8
0
2
PAD_INFO
70
2
10
0.000000
20
0.000000
0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
2
2
CDN_TYPE
```

```
SDA_1112
1
PAD_INFO
3


0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_LAYER
1
0
3
Pad layer?
0
ATTDEF
8
0
10
```

```
0.000000
20
0.000000
40
0.000000
70
0
2
SHAPE_TYPE
1
CIRCLE
3
Shape type?
0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_TYPE
```

```
1
REGULAR
3
Pad type?
0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_WIDTH
1
0.0
3
Pad width?
0
ATTDEF
8
0
10
```

```
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_HEIGHT
1
0.0
3
Pad height?
0
ENDBLK
0
BLOCK
8
0
2
DRILL_INFO
70
2
10
0.000000
20
0.000000
0
```

```
0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
2
2
CDN_TYPE
1
DRILL_INFO
3


0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
```

```
70
0
2
DIAMETER
1


3
Drill diameter?
0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PLATED
1


3
Plated?
0
```

```
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
FSHAPE
1


3
Fshape?
0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
```

```
70
0
2
FCHAR
1


3
Fchar?
0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
DRILL_WIDTH
1


3
Drill width?
0
ATTDEF
```

```
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
DRILL_HEIGHT
1


3
Drill height?
0
ENDBLK
0
BLOCK
8
0
2
SMD50_87
70
2
10
```

```
0.000000
20
0.000000
0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
CDN_TYPE
1
VIA
3
Pin or Via?
0
ATTDEF
8
0
10
0.000000
```

```
20
0.000000
40
0.000000
70
2
2
PSTKNAME
1
SMD50_87
3


0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PIN
1
```

```
0
3
Pin num?
0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
2
2
BBV
1
BLIND
3


0
INSERT
8
0
2
PAD_INFO
```

```
10
0.000000
20
0.000000
50
0.000000
66
1
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_LAYER
1
ETCH_TOP
0
ATTRIB
8
0
```

```
10
0.000000
20
0.000000
40
0.000000
70
0
2
SHAPE_TYPE
1
NULL
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_TYPE
1
```

```
ANTIPAD
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_WIDTH
1
0.000000
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
```

```
70
0
2
PAD_HEIGHT
1
0.000000
0
SEQEND
0
INSERT
8
0
2
PAD_INFO
10
0.000000
20
0.000000
50
0.000000
66
1
0
ATTRIB
8
0
10
```

```
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_LAYER
1
ETCH_TOP
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
SHAPE_TYPE
1
NULL
0
```

```
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_TYPE
1
THERMAL
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
```

```
2
PAD_WIDTH
1
0.000000
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_HEIGHT
1
0.000000
0
SEQEND
0
INSERT
8
0
2
```

```
PAD_INFO
10
0.000000
20
0.000000
50
0.000000
66
1
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_LAYER
1
ETCH_TOP
0
ATTRIB
```

```
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
SHAPE_TYPE
1
RECTANGLE
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_TYPE
```

```
PAD_TYPE
1
REGULAR
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_WIDTH
1
5000.000000
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
```

```
0.000000
70
0
2
PAD_HEIGHT
1
8700.000000
0
SEQEND
0
SOLID
8
ETCH_TOP
10
-0.025000
20
-0.043500
11
0.025000
21
-0.043500
12
-0.025000
22
0.043500
13
0.025000
```

```
23
0.043500
0
INSERT
8
0
2
PAD_INFO
10
0.000000
20
0.000000
50
0.000000
66
1
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
```

```
0
2
PAD_LAYER
1
ETCH_BOTTOM
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
SHAPE_TYPE
1
NULL
0
ATTRIB
8
0
10
0.000000
```

```
20
0.000000
40
0.000000
70
0
2
PAD_TYPE
1
ANTIPAD
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_WIDTH
1
0.000000
0
ATTRIB
```

```
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_HEIGHT
1
0.000000
0
SEQEND
0
INSERT
8
0
2
PAD_INFO
10
0.000000
20
0.000000
50
```

```
0.000000
66
1
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_LAYER
1
ETCH_BOTTOM
0
ATTRIB
8
0
10
0.000000
20
0.000000
```

```
40
0.000000
70
0
2
SHAPE_TYPE
1
NULL
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_TYPE
1
THERMAL
0
ATTRIB
8
```

```
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_WIDTH
1
0.000000
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_HEIGHT
1
```

```
-
0.000000
0
SEQEND
0
INSERT
8
0
2
PAD_INFO
10
0.000000
20
0.000000
50
0.000000
66
1
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
```

```
70
0
2
PAD_LAYER
1
ETCH_BOTTOM
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
SHAPE_TYPE
1
NULL
0
ATTRIB
8
0
10
```

```
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_TYPE
1
REGULAR
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_WIDTH
1
0.000000
```

```
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
PAD_HEIGHT
1
0.000000
0
SEQEND
0
INSERT
8
0
2
DRILL_INFO
10
0.000000
20
0.000000
```

```
0.000000
50
0.000000
66
1
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
DIAMETER
1
0.036000
0
ATTRIB
8
0
10
0.000000
20
```

```
0.000000
40
0.000000
70
0
2
PLATED
1
PLATED
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
FSHAPE
1
NULL
0
ATTRIB
```

```
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
FCHAR
1


0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
```

```
DRILL_WIDTH
1
0.000000
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
DRILL_HEIGHT
1
0.000000
0
SEQEND
0
ENDBLK
0
BLOCK
8
0
```

```
2
SMDCAP
70
0
10
0.000000
20
0.000000
0
ATTDEF
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
CDN_TYPE
1
PACKAGE
3


0
ATTDEF
```

```
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
REFDES
1
*
3
Refdes?
0
INSERT
8
0
2
SMD50_87
10
-0.100000
20
0.000000
50
```

```
0.000000
66
1
0
SEQEND
0
INSERT
8
0
2
SMD50_87
10
0.095000
20
0.000000
50
0.000000
66
1
0
SEQEND
0
POLYLINE
8
PACKAGE GEOMETRY_SILKSCREEN_TOP
66
1
```

```
70
1
40
0.000000
41
0.000000
0
VERTEX
8
PACKAGE GEOMETRY_SILKSCREEN_TOP
10
-0.075000
20
-0.045000
40
0.000000
41
0.000000
42
0.000000
0
VERTEX
8
PACKAGE GEOMETRY_SILKSCREEN_TOP
10
-0.075000
20
```

```
0.045000
40
0.000000
41
0.000000
42
0.000000
0
VERTEX
8
PACKAGE GEOMETRY_SILKSCREEN_TOP
10
0.070000
20
0.045000
40
0.000000
41
0.000000
42
0.000000
0
VERTEX
8
PACKAGE GEOMETRY_SILKSCREEN_TOP
10
0.070000
20
```

```
-0.045000
40
0.000000
41
0.000000
42
0.000000
0
VERTEX
8
PACKAGE GEOMETRY_SILKSCREEN_TOP
10
-0.075000
20
-0.045000
40
0.000000
41
0.000000
42
0.000000
0
SEQEND
0
POLYLINE
8
PACKAGE GEOMETRY_ASSEMBLY_TOP
```

```
66
1
70
1
40
0.000000
41
0.000000
0
VERTEX
8
PACKAGE GEOMETRY_ASSEMBLY_TOP
10
-0.075000
20
-0.045000
40
0.000000
41
0.000000
42
0.000000
0
VERTEX
8
PACKAGE GEOMETRY_ASSEMBLY_TOP
10
```

```
-0.075000
20
0.045000
40
0.000000
41
0.000000
42
0.000000
0
VERTEX
8
PACKAGE GEOMETRY_ASSEMBLY_TOP
10
0.070000
20
0.045000
40
0.000000
41
0.000000
42
0.000000
0
VERTEX
8
PACKAGE GEOMETRY_ASSEMBLY_TOP
```

```
10
0.070000
20
-0.045000
40
0.000000
41
0.000000
42
0.000000
0
VERTEX
8
PACKAGE GEOMETRY_ASSEMBLY_TOP
10
-0.075000
20
-0.045000
40
0.000000
41
0.000000
42
0.000000
0
SEQEND
0
ENDBLK
```

```
ENDBLK
0
ENDSEC
0
SECTION
2
ENTITIES
0
INSERT
8
0
2
SMDCAP
10
1.710000
20
3.640000
50
270.000000
66
1
0
ATTRIB
8
0
10
0.000000
20
```

```
0.000000
40
0.000000
70
0
2
SYMBOL_TYPE
1
PACKAGE
0
ATTRIB
8
0
10
0.000000
20
0.000000
40
0.000000
70
0
2
REFDES
1
*
0
SEQEND
```

```
0
ENDSEC
0
EOF
```