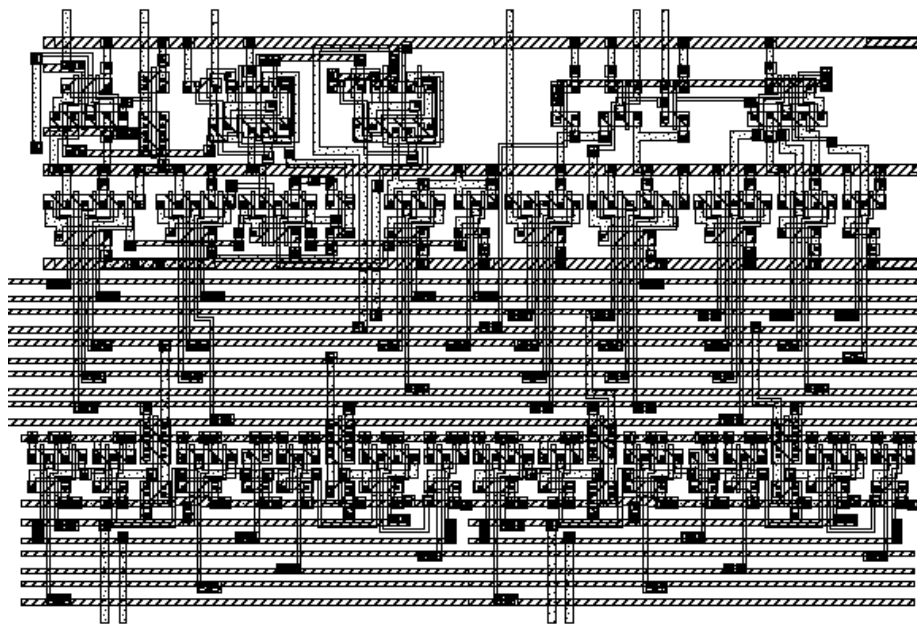


ICEDTM

IC Layout Editor

Command and Utility Reference



Updated for Version 4.82

IC Editors, Inc.

© 2004 by IC Editors, Inc.

No part of the information contained in this manual may be represented in any form without the prior written consent of IC Editors, Inc.

Limited Warranty

IC Editors, Inc. warrants that the program will substantially conform to the published specifications and to the documentation, provided it is used on the computer hardware and with the operating system for which it is designed. IC Editors, Inc. also warrants the media on which the program is distributed and the documentation are free from defects in materials and workmanship.

IC Editors, Inc. will replace defective documentation or correct substantial program errors at no charge, provided you return the item to IC Editors, Inc. within 1 year of the date of delivery. IC Editors, Inc. will replace defective media at no charge provided you return the item to IC Editors, Inc. within 3 years of the date of delivery. If IC Editors, Inc. is unable to replace defective media or documentation or correct substantial program errors, IC Editors, Inc. will refund the purchase payment for the product. These are your sole remedies for any breach of warranty.

Except as specifically provided above, IC Editors, Inc. makes no warranty or representation, either express or implied, with respect to this program or documentation, including their quality, performance, merchantability, or fitness for a particular purpose.

Because programs are inherently complex and may not be completely free of errors, you are advised to validate your work. In no event will IC Editors, Inc. be liable for direct, indirect, special, incidental, or consequential damages arising out of the use of or inability to use the program or documentation, even if advised of the possibility of such damages. Specifically, IC Editors, Inc. is not responsible for any costs including but not limited to those incurred as a result of lost profits or revenue, loss of use of computer program, loss of data, the costs of recovering such programs or data, the cost of any substitute program, claims by third parties, or other similar costs. In no case shall IC Editors, Inc.'s liability exceed the amount of the purchase payment.

The warranty and remedies set forth above are exclusive and in lieu of all other, oral or written, express or implied. No IC Editors, Inc. dealer, distributor, agent, or employee is authorized to make any modification or addition to this warranty. Some states do not allow the exclusion or limitation of implied warranties or limitation of liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Software License

The ICED™ software is protected by United States copyright law and international treaty provision.

While the software may be installed on multiple systems, the use of the software is restricted to individual systems using hardware keys supplied by IC Editors, Inc. Interference with the function of the hardware keys is prohibited.

The sale or transfer of the software to a third party is prohibited without the prior permission IC Editors, Inc.

Acknowledgments

The majority of this manual was written or revised by Ference Professional Services in Sonora, CA. We are also responsible for formatting the text and creating the screen captures that illustrate the examples. The original ICED-16 manual was written by Mark Stegall. Pieces of this manual can still be found in this ICED™ manual.

Michael Gentry of MGC, Inc. created the layout that is used on the cover and as a frontispiece. It is a section of a CMOS simulation of a 74181 4-bit ALU.

Table of Contents

TABLE OF CONTENTS	1
INTRODUCTION	7
OVERVIEW OF THE ICED™ ENVIRONMENT	11
ICED™ Programs.....	12
Major Types of ICED™ Files	13
ICED™ Directories and Search Paths	16
Opening the Layout Editor	23
Steps to Take Before Creating Cells in a New Project	24
STARTUP COMMAND FILES	25
Sample NEW.CMD	26
Creating a New Startup Command File	28
Environment Settings in Startup Command Files	29
Always Command Files	35
Hierarchical Startup Command Files.....	36
Changing a Startup Command File	36
PROJECT BATCH FILES	39
Sample ICWIN.BAT	40
Creating a New Project Batch File	41
Defining Environment Variables	43
ICED.EXE Command Line Syntax	51
Command Line Parameters.....	54
ICED™ Command and Utility Reference	1

Table of Contents

Memory Management.....	87
Executing Batch Files on a Windows Platform	90
Using ICED™ to Perform Tasks with no User Interaction.....	91

OVERVIEW OF THE ICED™ LAYOUT EDITOR 93

The ICED™ Layout Editor Window.....	94
Components and Cells	94
Geometric Database vs. Environment Database	95
Creating New Cells.....	96
Opening Existing Cells.....	97
When Cell Data Is Saved in Cell Files.....	98
Commands and Command Files	98
Journaling	99
Terminating ICED™	100

SYNTAX CONVENTIONS 101

Manual Notation.....	102
Layer Lists.....	106

ICED™ COMMAND REFERENCE..... 109

Overview	110
<i>@file_name</i> Execute a command file.....	115
<i>\$comment</i> Add comment to journal file and screen.	120
ADD Add a component to the drawing	122
ARRAY Set the array display mode.	151
ARROW Change function of arrow keys.	153
AUTOPAN Enable or disable automatic view panning.	156
BLANK and UNBLANK Hide or show layers or components.....	158
BLINK Make the indicated color blink on the screen.	163
CIF Output design as a CIF format file.....	165
COLOR Set colors and priority levels.	167
COPY Copy existing components.	175
CURSOR Set position cursor type.	179

CUT	Cut a box, polygon, wire, or line into two parts.....	180
DELETE	Remove all fully selected components from the drawing.	181
DISPLAY	Control how ICED™ displays data.	182
DOS	Launch console application and wait.....	184
DRC	Create an intermediate file for the DRC program.....	188
EDIT	Edit another cell.....	192
EXIT	Terminate the edit session and save the cell.	198
FILL	Turn component fill patterns ON or OFF.	201
GRID	Define display grid parameters.....	203
GROUP and UNGROUP	Group selected components into a cell.	206
INITIALIZE	Set the parameters of a list of layers.	210
JOURNAL	Terminate ICED™ without saving work.	211
KEY	Assign a command string macro to a function key.	213
LAYER	Set the default layer or layer parameters.	216
LEAVE	Terminate the edit session and save the cell if it has changed.....	229
LIST	Save a named list of components.....	231
LOG	Speed command files by controlling how commands are logged.	234
MENU	Load a menu file.....	241
MERGE	Merge wires or lines. Merge or subtract polygons.	243
MIRROR	Move components by mirroring about an axis.....	248
MOVE	Move existing components or their vertices.	250
NEAR	Set the half width of the near box.....	255
OUTLINE	Set depth for which cell and array outlines are drawn.	256
P_EDIT	Edit another cell in place.....	258
PACK	Compress buffer space; report on memory usage.....	262
PATTERN	Load a stipple pattern file.	263
PAUSE	Create a pause in a command file.....	265
PLOT	Create an intermediate file for the MkPlot utility.....	267
PROTECT and UNPROTECT	Prevent changes to components or layers.	284
QUIT	Terminate the edit session without saving the current cell.....	287
REDRAW	Refresh the screen.....	289
RESOLUTION	Define the grid for digitizing points.....	290
ROTATE	Move components by rotating about a point.....	292
RULER	Measure the distance between two points.	294

Table of Contents

SELECT and UNSELECT	Select or unselect components.....	295
SHOW	Display component or macro data.....	307
SNAP	Define a temporary grid for digitizing coordinate data.....	318
SPACER	Modify spacing cursor for adding wires or polygons.....	322
SPAWN	Launch console application without a wait.....	329
STREAM	Output design as a Stream (Calma-GDSII compatible) format file.....	333
SWAP	Swap layers of components or replace cells.....	338
T_EDIT	Edit another cell with a transformed coordinate system.....	340
TEMPLATE	Display or save template parameters.....	344
TEXT	Set parameters that affect how text components are added and displayed.....	346
UNDO	Reverse effects of the previous command.....	350
USE	Set default values for parameters used by the ADD command.....	351
VIEW	Set the view window.....	354
VIEW (ON OFF)	Control display refresh during execution of command files.....	360
VIEW LIMIT	Speed screen refresh by limiting display detail.....	363
XSELECT	Enable or disable embedded selects in command files.....	367

ICED™ UTILITY REFERENCE 369

Overview	370
Executing ICED™ Utilities	371
AllCells	Execute a command file on all cells in a library.....378
Del2Cel	Convert @...DEL files to .CEL files.382
ICList and ICTree	List cells nested in a cell file.....384
ICTop	Find possible top-level cells.....388
MkMenu	Create custom menus.....391
MkPDF	Modify a PDF file.393
MkPlot	Print or plot .VEC plot files.396
MkSti and UnMkSti	Prepare stipple file for fill pattern creation.....402
SFDmp	Create an ASCII dump from a GDSII Stream Format file.405
SFMap	Change GDSII Stream file structure names.....407
UnCIF	Convert CIF files into ICED™ cell files.412
UnStream	Convert GDSII Stream Format files into ICED™ cell files.418

APPENDIX A: ICED™ FILES.....	431
ICED™ File Types.....	432
One File That Should Not Be Deleted Or Copied	438
 APPENDIX B: JOURNALING AND DATA RECOVERY	 439
How Cell Files are Saved	440
The Journal File.....	441
Journaling During Command Files	442
Automatic Data Recovery.....	442
Recovering From Mistakes.....	444
Recovery After Cell Files Are Saved.....	445
 APPENDIX C: ADVANCED BATCH FILE EXAMPLES	 449
Creating a Desktop Shortcut to Plot a Cell	451
Creating Batch Files for Repeatable Procedures	456
 INDEX.....	 461

Introduction

.	DELETE
.	VIEW
.	in %
.	out %
.	box
.	last
.	all
.	COPY
.	MOVE
.	side
.	UNDO
.	Again
.	UseLay
.	SELECT
.	layer
.	new
.	in
.	side
.	ADD
.	box
.	poly
.	wire
.	cell
.	text
.	USE
WELL: Sel=0; X> █	FILE
GLOBAL #KEY.F9="DOS"	

ICED™ is a full-featured layout editor. It can create and manipulate all of the design features used in integrated circuit mask sets. In addition, it can import and export data in CIF (Caltech Intermediate Form) and Stream (Calma-GDSII) formats. Other programs available from IC Editors, Inc. provide design rule checking and verify the correspondence between the layout and the schematic.

See the "Classroom Tutorials" manual for a hands-on approach to learning ICED™. This manual can be downloaded from the website WWW-ICEDITORS.COM

The "Command File Programmer's Reference Manual" contains details on advanced features to automate complex layout tasks.

This manual is intended to be a complete reference guide for all features of the ICED™ editor (except for the macro programming language that is described in the Command File Programmer's Reference Manual). This manual covers the complete syntax of the commands used inside the editor, and the stand alone utilities that can be used to create and report on ICED™ data.

This manual is a reference manual and is intended primarily for users who are already somewhat familiar with the layout editor. If you are a new user, you many want to read the basic tutorial in the "Classroom Tutorials" manual before continuing.

The main sections of this manual are described below.

"Overview of the ICED™ Environment" describes ICED™ files and how customizations like the cell library search path and technology settings are passed into the layout editor. In addition, the steps to open the editor are introduced.

"Startup Command Files" covers the details on how technology and editor settings are defined for new cells.

"Project Batch Files" explains how to customize a batch file to launch ICED™. This chapter includes an explanation of the required environment variables and the layout editor command line. A detailed explanation of each of the ICED™ command line parameters is included.

"Overview of the ICED™ Layout Editor" provides an overview of how the layout editor creates, manipulates and stores data. This chapter introduces important terminology used to describe the ICED™ cell database. In addition, the process of journaling, and how ICED™ saves cell files is described in this section.

"Syntax Conventions" explains the notation used to describe the syntax of the commands used inside the ICED™ layout editor. It also includes an explanation of layer lists.

The **"ICED™ Command Reference"** contains the syntax of every command used inside the editor along with examples. This chapter starts with an overview that provides a brief description of each command and its purpose.

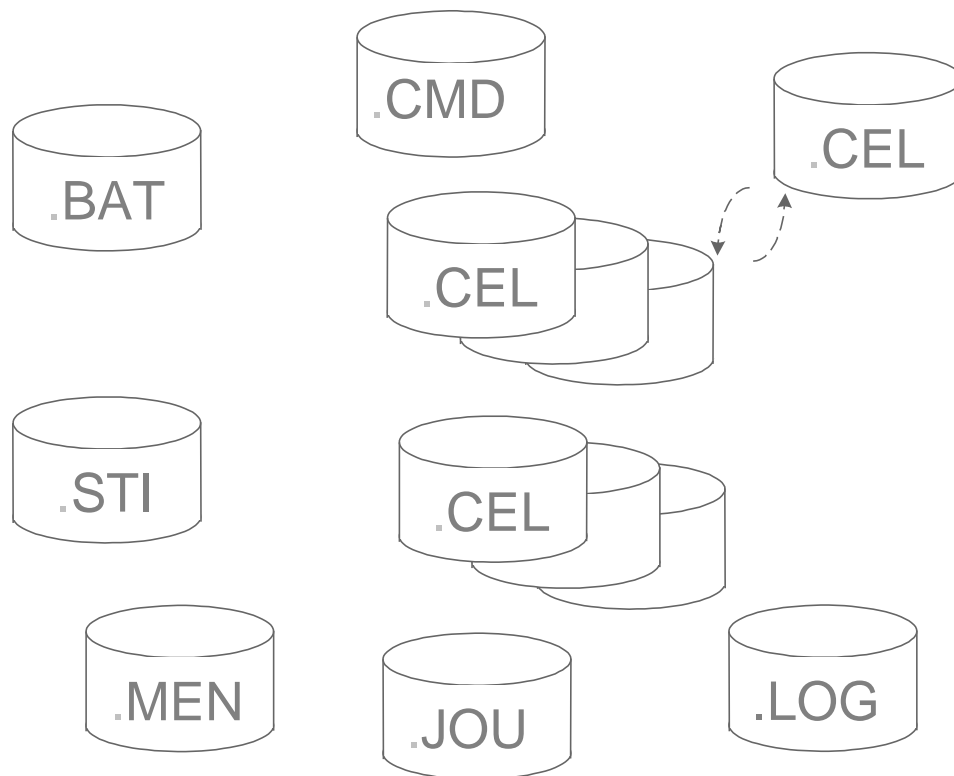
The **"ICED™ Utility Reference"** describes a number of utility programs that come with ICED™. These programs are run from the console prompt and perform a variety of tasks including design import. See the chapter overview for a brief description of each utility.

Appendix A covers the various files associated with ICED™ in more detail.

Appendix B describes the data recovery options available to you in the event of a system crash or severe error. Several recovery scenarios are covered. Using the more advanced examples requires an understanding of journal files and cell backup files which are introduced in the "Overview of the ICED™ Layout Editor" chapter.

Appendix C illustrates two examples of how to launch ICED™ with a batch file to perform automated tasks. One example plots a chip by clicking a button on your desktop, the other exports data for the DRC (Design Rules Checker, a separate product available from IC Editors, Inc.)

Overview of the ICED™ Environment



ICED™ Programs

The syntax of the ICED.EXE command line is covered beginning on page 51.

The layout editor is just one of the executable programs in the ICED™ installation.

ICED.EXE

This program is the layout editor. Use this program to create, display, manipulate, and store layout geometry in individual cells. This program also exports layout data in several formats.

Modifying ICWIN.BAT is covered beginning on page 39.

ICWIN.BAT

This sample batch file contains the ICED.EXE command line. Execute this batch file, or a modified copy of it to launch the editor. The file includes the definition of several environment variables required by the editor.

All utilities are described in detail beginning on page 369.

Several other executable files are included in the basic installation. These utility programs perform operations such as auxiliary file compilation (e.g. the preparation of fill pattern files), data import (e.g. stream file translation), and the reporting of information (e.g. listing the cell hierarchy of a design).

Other important programs that manipulate ICED™ data can be purchased separately from IC Editors, Inc. These products include:

DRC.EXE

Design Rules Checker – used to verify that your layout meets technology-dependant design restrictions.

The DRC, NLE, and LVS programs are documented in separate manuals.

NLE.EXE

Net List Extractor – used to translate layout data into a list of device and net connections.

LVS.EXE

Layout Vs. Schematic – used to compare a netlist extracted from your layout to your schematic netlist. This verifies that the circuits in the physical data correspond to the circuits in the design schematics.

The ICED™ layout editor and related utilities are console based applications, which can be launched directly by typing in a MS-DOS console window, but they are not DOS applications and they cannot be used on a pure DOS platform. They take advantage of Windows multitasking and virtual memory capabilities.

Major Types of ICED™ Files

The information below is a brief overview of the major types of files in the ICED™ environment so that they are fresh in your mind before we discuss where different types of files are stored and how their search paths are defined.

Cell Files

Layout geometry is stored in cell files. These files are designed to optimize speed and storage space so they are not written in ASCII and can be viewed or edited only with the ICED™ layout editor.

A more complete description of cells, subcells, and cell files is found on page 94.

A cell is basically a collection of geometric shapes. Cells can contain references to other cells, called subcells.

Each cell is stored as a unit in its own cell file. The name of a cell file is *cell_name.CEL*, where *cell_name* is the name of the cell. (Cell backups use the extension ".CL1".)

When ICED™ must search for cell files, you must define a cell library search path. This is usually done in the same batch file as the ICED.EXE command line.

Batch Files

A more complete description of project batch files is found beginning on page 39.

The environment for the layout editor (e.g. the cell library search path) must be defined with operating system commands before executing the ICED.EXE program. Typically these operating system commands are combined with the ICED.EXE command line in a batch file. A copy of the sample batch supplied with the installation, Q:\ICWIN¹\ICWIN.BAT, can be customized to refer to your technology files and cell libraries. This customized batch file is referred to as a **project batch file**.

A batch file can be created with any ASCII text editor. The file extension should be .BAT. Avoid blanks in the file name.

Journal Files

A more complete description of journaling is found on page 99.

As the layout editor executes each command, the command is logged into a journal file. Since commands are stored in the journal file, it can be used as a command file. This journal file has several uses including automatic recovery of layout data in the event of a system crash or editing mistake.

This journal file is written out to disk frequently during a layout editor session. The name of the file while it is being written is *cell_name*.JOU. If your editor session is terminated normally, this file will be renamed to *cell_name*.LOG. (These files can also have file extensions of .JO1 and .JOX when used for recovery.)

Command Files

See page 98 to learn how to execute command files.

The layout editor is command-driven. These commands can be executed by typing them on the command line, by selecting commands from a menu, or executing a command file. Command files are created by collecting groups of

¹ Throughout this manual, Q:\ICWIN represents the drive letter and path where you have installed ICED™. Replace "Q:" with the drive letter assigned during installation and "ICWIN" with the directory name created at that time.

commands in a file using any ASCII editor. When the command file is executed, the commands in it are executed.

The "Command File Programmer's Reference Manual"

contains a lot of information not covered in this manual that can make command files more powerful.

Command files can be extremely powerful, since they can contain user prompts, conditional statements, and mathematical expressions. These files can also be very simple lists of commands that insure that the same sequence of commands is executed each time.

Command files can be divided into two different classes:

- General purpose command files that can be performed in any cell, independent of the specific technology in use, or
- Technology-dependant and project-dependant command files that contain values or operations specific to only certain designs.

Learn about the command file search path on page 45.

We will later discuss the value of keeping these two types of command files separate.

Command files can have any file name and file extension. However, we recommend the following restrictions to avoid problems:

- Blanks in file names (or directory names) can cause syntax problems for the command parser unless the entire file name is surrounded with quotes.
- Only files with a .CMD extension will added to the menus for easy selection with the mouse.

Command files with names that begin with a " " prefix will not be added to the menu lists.

Many sample command files are distributed with the program. Q:\ICWIN²\AUXIL\DEEPSHOW.CMD can be used from any cell to report on geometry nested in a subcell. It is an example of a general purpose command file. Q:\ICWIN\TECH\SAMPLES\NEW.CMD is a good example of a technology-dependent command file. This file is a sample of a startup command file that defines technology settings for each new cell.

² Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

Startup Command Files

The name of the startup command file is specified with the STARTUP option on the ICED.EXE command line in the project batch file.

When a **new cell** is created, ICED™ will execute the commands in a special command file that initializes **technology settings** such as layer properties and grid resolutions. It also defines and **layout editor settings** like keyboard shortcuts. Settings like these are collectively referred to as the **environment of a cell**. The cell environment is stored with the cell file.

One of the most important tasks for a new ICED™ user is the preparation of a startup command file for their project. We will cover this subject in detail beginning on page 25. The sample startup command file mentioned above, Q:\ICWIN\TECH\SAMPLES\NEW.CMD, will be fully explained here.

Auxiliary Files

Learn about how to modify the auxiliary file search path on page 44.

Support files such as **menu** and **pattern** files are stored in machine-readable format. These files are usually stored together in the Q:\ICWIN\AUXIL directory.

ICED™ Directories and Search Paths

Most ICED™ files are found by the program by looking in various search paths. These search paths are stored in operating system environment variables defined before you open the layout editor. These definitions are usually made in the same project batch file that contains the ICED.EXE layout editor command line.

We devote an entire chapter to how to create a project batch file beginning on page 39. It will help make this explanation easier if you are familiar with the directory structure created by the installation and the types of search paths that should be defined.

Installation Directories

The directory name you specified as the location of the ICED™ program is only the root directory of several subdirectories created by the installation. Each of these subdirectories contains different types of files. The directories created by the installation are explained in the table below.

The TUTOR directory is a good place to store cell files used only for experimentation. However, it is best to create new directories for your design cell files rather than store them in any of the other existing directories.

Directory		Contents
Subdirectories	Q:\ICWIN\	Executable files: the ICED.EXE file, utilities, and project batch files.
	AUXIL\	Technology-independent support files such as menus, pattern files, printer definition files, and general purpose command files.
	AZTECH\	Files used during installation of software protection key. Users usually do not need to access this directory.
	DOC\	ASCII text and Adobe Acrobat PDF files about ICED™ subjects. This information may occasionally be more up-to-date than this manual.
	PGTEXT\	Auxiliary file for maskable text routines. (See Classroom Tutorials Manual.)
	SAMPLES\	Sample files such as cell files.
	SETTINGS\	Installation files and files created by the editor to save user settings such as printer selection. Users usually do not need to access this directory.
	TECH\	Technology directories. No files are usually stored directly in the TECH\ directory, but each technology you use should have its own subdirectory stored here that contains mostly technology-dependant command files.
	TMP\	Temporary files such as swap files or scratch files. ³
	TUTOR\	A sample working directory for storing cell files created for experimentation with the layout editor. The Classroom Tutorials Manual uses this directory.

Figure 1: ICED™ directory structure

³ These temporary files can get very large. If the installation directory is not located on a local fixed drive with plenty of free space, see the alternate method of assigning a TMP directory on page 84.

Cell Libraries

A cell library is a directory that contains cell files. (Remember that each cell is saved in a separate cell file.) It is best to separate directories that contain cell files from the directories created by the ICED™ installation.

Avoid blanks in the directory names of cell libraries. If blanks are present in any of the directories on the library path, you will have to use quotes around the library name to avoid syntax problems.

There are no practical limitations on how to organize your cell library directories. A cell library can be located in any convenient directory. You can separate cell files into different cell libraries by any classification. You can refer to as many different cell libraries in a given project as required. Some of these libraries can be located on a shared network drive while others are on your local drive or even on a CD-ROM drive.

The only directories that should be avoided for use as cell libraries are the directories created by the ICED™ installation. For example, do not store cell files in the Q:\ICWIN\AUXIL directory. This combination of user cell files with editor support files would lead to unnecessary confusion. However, creating a new subdirectory of the installation directory (e.g. Q:\ICWIN\CELLS\MYLIB) is acceptable and often useful.

For more details on cell library definition with ICED_PATH, see page 45.

This flexibility in cell library definition is possible because each project has its own list of cell libraries. This list is defined by the system environment variable **ICED_PATH**.

The order in which the cell libraries are defined is important. The list stored in ICED_PATH determines the search order used by the layout editor. Cell files in libraries defined at the beginning of the cell library list override cell files with the same name in libraries closer to the end of the list.

Cell libraries can be protected from changes. Settings in the ICED_PATH associate a protection class for each cell library. The protection classes are:

- View-only** cells cannot be modified
- Copy-edit** cells can be modified, but a modified cell does not overwrite the original cell
- Direct-edit** a modified cell overwrites the original version

Details on how to assign cell library protection are provided on page 45.

The Working Directory

The current directory is the working directory unless you specify a directory path with *cell_name* on the ICED-.EXE command line.

The working directory is not searched for auxiliary files such as menu files.

The journal file is written to frequently in an edit session. If the working directory is not located on a local fixed disk drive, it will slow down your edit session considerably.

There is a special cell library treated differently from the others in every ICED™ session. This is the working directory.

- The working directory is always the first directory on following search paths:
 - ◆ the cell library search path,
 - ◆ the command file search path, and
 - ◆ the executable file search path.
- The working directory is the library from which you select the **root cell**, i.e. the first layout drawing you open with the editor.
- New cells created during the edit session will be stored in the working directory.
- When you modify a cell from a copy-edit cell library, the modified cell is stored in the working directory and the original cell in the copy-edit library is unchanged.
- When you export data from a cell in the working directory or in a protected cell library, the exported file is stored in the working directory. (This applies to intermediate files used for plotting and DRC checking as well as files exported using the STREAM or CIF commands.)
- The journal file (the file that stores each command executed during the edit session) is stored in the working directory.

Subcells nested inside of a cell being edited by the layout editor must be located in the working directory or in one of the cell libraries listed in the ICED_PATH. If the cell file for a subcell cannot be found in these directories, the editor will not be able to open the cell containing the subcell.

When ICED™ searches for cells or subcells, it will search for the cell file first in the working directory, then through all directories defined as cell libraries in the order that they are defined in the ICED_PATH. This means that if two or more cells with the same name exist in more than one cell library, the cell from the first library on the list of libraries will be used, and other copies will be ignored. Which copy of the cell will be used in a later ICED™ session depends on the current working directory and the order of the cell libraries in the current project.

Since files must be written to the working directory, you should not select the following types of directories for your working directory:

- A directory that is on a read-only drive (e.g. a CD-ROM drive).
- A directory that is defined as a protected cell library in the current project.

One other type of directory is a poor choice for a working directory because it will slow down your edit session.

Avoid using
blanks in a
working
directory name.

- A directory that is on a removable drive (e.g. Zip drive) or a network drive. File creation on these drives is very slow and since the journal file is updated constantly, you will see a major increase in the time it takes to perform almost every command.

If you need to edit or view a cell in one of these types of directories, use the nested edit method outlined below.

Accessing Cells in Directories Other Than the Working Directory

When you cannot use a cell library as a working directory, but you need to edit or view a cell in the library, you must do this with a nested edit.

- Be sure that the library is defined in the list of cell libraries in the current project batch file. To edit the cell, the cell library cannot be a view-only library. (This is the default protection status. See page 45.)
- Select a working directory you do have access to, then open the layout editor to edit any cell in that directory as the root cell. Create a new dummy cell if necessary.
- Edit the cell in the cell library with the EDIT CELL command (or the 2:EDIT⁴→CELL menu option.) Alternatively, you can add the cell of interest to the dummy cell with the ADD CELL command, then use the P_EDIT or T_EDIT commands to edit the cell. In this case, the cell is displayed automatically the next time you open the dummy cell.

⁴ When we discuss menu options, the number before the colon refers to the top-level menu number. (You switch between top-level menus with the right mouse button.) The options after the → character are submenu options.

See page 45 to learn about protected libraries.

If the cell of interest is in a copy-edit cell library, the modified cell file will be stored in the working directory when you terminate the layout editor.

Location of Command Files

See page 14 for a definition of command files. See page 98 to learn how to execute command files.

General-purpose command files should be stored in the Q:\ICWIN\AUXIL subdirectory where they will be found automatically by the layout editor for all projects.

However, some command files are technology or project dependant. The most important example is the startup command file that initializes a variety of technology-dependant parameters such as layer number-layer name correspondences, resolution grid definitions, etc. You may have other command files that use contact or wire spacing distances or other settings that you want to use only in specific projects.

Learn more about the importance of the startup command file on page 25.

You should not keep these types of command files in the AUXIL subdirectory of the installation directory where they could be accidentally accessed in other projects that use other technologies. It would be too easy to execute such a command file in the wrong cell file, and you may not realize your mistake for a long time. If you realize this type of mistake after fabrication, you will be particularly displeased.

It is easy to prevent this type of error by locating all technology or project dependent command files in separate directories. These directories can then be added to the command file search path of a particular project in the project batch file. This search path is defined with the **ICED_CMD_PATH** environment variable we cover on page 48.

Older versions of ICED™ always searched the cell libraries for command files, but this long search path has the potential to add a large overhead to the search of network drives. So newer versions support this detailed but flexible search path to separate general purpose command files from technology-dependent command files, and command files from cell files.

Technology Directories

For the reason mentioned above, and also to simplify the process of migrating to new technologies, we strongly recommend storing technology-dependent command files in separate technology directories. Technology directories are usually stored as subdirectories of the Q:\ICWIN\TECH subdirectory. However, you can locate a technology directory on an appropriate shared drive if you need to share technology files with other users over a network.

You specify the name of the technology directory (or directories) in the command file search path stored in the ICED_CMD_PATH environment variable in the project batch file.

Avoid using blanks in a technology directory name.

See page 48 to get details on defining the ICED_CMD_PATH.

To support a new technology, you would do the following:

- create a new technology subdirectory, e.g. Q:\ICWIN\TECH\BIPOL,
- copy files as needed from an old technology directory (e.g. Q:\ICWIN\TECH\SAMPLES) to the new one,
- edit **all** of the copied command files in the new directory to use the new technology settings,
- and finally, edit the definition of ICED_CMD_PATH in your project batch file to use the new technology directory.

Auxiliary File Search Path

Auxiliary files such as menu, pattern, and printer files are technology-independent files that you will probably use in many different projects and technologies. So these files are usually stored together in the Q:\ICWIN\AUXIL directory. General purpose command files are also stored here and so this directory is automatically added to the command file search path.

However, when you need to share these types of files over a network, or if you want to keep the original files supplied with the installation intact while you use copies modified for your use, you may want to search more than one directory for auxiliary files. You can do this by modifying the ICED_HOME environment variable definition. We cover this in detail on page 44.

Opening the Layout Editor

The Q:\ICWIN directory is added to the system executable search path automatically. See page 49.

The usual method to open the layout editor is as follows:

- Open a console window with the ICED icon on your desktop created by the installation. The Q:\ICWIN⁵ directory is the current directory in this new console window.
- Make your working directory the current directory with the CD MS-DOS command. (If you need to change the current drive as well, you do this first with by typing the drive letter followed by a ':'.)
- Execute the project batch file with the root cell name as an argument.
- If the root cell file does not yet exist, the startup command file specified in the project batch file is executed to initialize editor settings.



Figure 2:
ICED
desktop
icon

For example, you could type the following commands at the prompt in a new open console window to open the layout editor to edit the cell MYCELL in the Q:\ICWIN\TUTOR directory:

Example:

**CD TUTOR
ICWIN MYCELL**

Some types of directories should not be used as working directories. See page 19.

The contents of ICWIN.BAT are shown on page 40.

The first command makes the Q:\ICWIN\TUTOR directory the current directory. The current directory is used as the working directory (unless you specify a working directory path with the *cell_name* specification.)

The second command executes the sample project batch file, Q:\ICWIN\ICWIN.BAT. (This executable file can be found by the system because the console window opened by the ICED icon has the Q:\ICWIN directory automatically added to the system executable search path.) The batch file will be executed with the string MYCELL as an argument. This will result in opening the layout editor to edit the root cell MYCELL after the batch file defines several environment variables that set various search paths.

⁵ Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

The contents of NEW.CMD are shown on page 28.

If the cell file MYCELL.CEL does not yet exist in the Q:\ICWIN\TUTOR directory, the startup command file specified in the sample project batch file (Q:\ICWIN\TECH\SAMPLES\NEW.CMD) is executed. Once it is complete, the layout editor window is displayed.

To close the editor and save the cell file, you type the EXIT command.

Steps to Take Before Creating Cells in a New Project

- Create a new technology directory, usually a new subdirectory of Q:\ICWIN\TECH. (See page 22.)
- Create a startup command file for your technology, usually a copy of Q:\ICWIN\TECH\SAMPLES\NEW.CMD. Customize it for your technology and store the file in the new technology directory. Details on this are covered in the next chapter.
- Create a project batch file, usually a copy of Q:\ICWIN\ICWIN.BAT. Customize the search paths and ICED.EXE command line options and store the file in the Q:\ICWIN directory. Details on project batch files are covered beginning on page 39.
- Create a new empty directory for your working directory cell library. Execute the project batch file while this directory is the current directory to open the layout editor.

Startup Command Files

```
COLOR RED    PALETTE=(63, 0,21)    LEVEL= 8
COLOR CYAN  ONE_DOT=CYAN  FOUR_DOT=(CYAN, CYAN, CYAN, CYAN)
GRID 1      ONCOLOR=RED  DOTS      STEP=1.0
LAYER 0  PEN=0
LAYER 1  NAME=NWEL  WIDTH=3.000  SPACE=0.0  BLUE  PAT=0  PEN=16
LAYER 2  NAME=NDIF  WIDTH=3.000  SPACE=0.0  GREENPAT=1  PEN=*
GLOBAL KEY.CF9="@UNED"
GLOBAL KEY.F1="RULER"
```

The STARTUP option is used to assign the startup command file on the ICED.EXE command line in the project batch file.

You can create a command file with all cell environment settings from an existing cell with the TEMPLATE command in the layout editor.

We have already mentioned the importance of a startup command file. This file contains ICED™ editor commands that initialize the environment and technology settings of a **new cell**. The startup command file serves the same purpose as the technology files used in many other systems.

It is critical to prepare a good startup command file **before any cells are created**. This ensures that all cells for a given project have the same technology parameters such as grid definitions, patterns, and layer names.

You will generally need a different startup command file for each process. We suggest that you keep the startup command file and all other technology-dependant command files together in an appropriately named subdirectory of the Q:\ICWIN\TECH directory. Add this directory to the command file search path only for projects that use this technology. (See page 48.) We recommend that you **do not** copy startup command files into your working directories or cell libraries. (This can result in different technology settings in different cells.)

Sample NEW.CMD

The easiest way to create your own startup command file is to make a copy of the sample file supplied with ICED™, Q:\ICWIN⁶\TECH\SAMPLES\NEW.CMD, and modify it with a text editor. The contents of this file are shown on the next page. All of these statements use commands described in more detail in the command reference section of this manual.

⁶ Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

```

VIEW      OFF
$ MENU    "M1";      KEEP_LIBRARY_CELLS=ASK;    DISPLAY CELL_DEPTH=100;
PATTERN   "SAMPLE";  FILL MIXED ON
AUTOPAN   ON PIXELS=100    SECONDS=0.5;  ARROW MODE=EDIT
DISPLAY   CELL_LABELS=ON   OUTLINE_DEPTH=1  EDIT_STACK=OFF  CURSOR=1 SNAP=ON
SPACER    OFF SPACE=0.0    TRACK_LAYERS=OFF  STYLE=1
VIEW LIMIT ON SCALE=0.500  DEPTH=1      DOTS=0      UNITS=0.0    SHOW_LAYERS 1:100
ARRAY     DRAW_MODE=SIDES  CELL_LIMIT=1024
TEXT      LOWER_CASE=DISABLED  MULTI_LINE_TEXT=DISABLED  ORIENTATIONS=2  &
          DISPLAY_ORIGINS=OFF
USE       TEXT_JUSTIFICATION=LB  WIRE_TYPE=2    ARC_TYPE=2  N_SIDES=16
RESOLUTION STEP=0.500  MODE=SOFT
SNAP      ANGLE=45     STEP=(0.500,0.500)    OFFSET=(0.000,0.000)
NEAR      UNITS=0.05   DOTS=4

COLOR 0   NAME=BLACK      PALETTE=( 0, 0, 0)
COLOR 1   NAME=WHITE      PALETTE=(63,63,63)  LEVEL=16
COLOR 2   NAME=YELLOW     PALETTE=(63,63, 0)  LEVEL= 6
COLOR 3   NAME=GREEN      PALETTE=(21,63, 0)  LEVEL= 6
COLOR 4   NAME=RED        PALETTE=(63, 0,21)  LEVEL= 8
COLOR 5   NAME=CYAN       PALETTE=( 0,42,42)  LEVEL= 9
COLOR 6   NAME=BLUE       PALETTE=( 0, 0,63)  LEVEL=10
COLOR 7   NAME=MAGENTA    PALETTE=(63, 0,63)  LEVEL= 8
COLOR 8   NAME=GRAY       PALETTE=(42,42,42)  LEVEL=14
COLOR 9   NAME=BROWN      PALETTE=(32,16, 0)  LEVEL= 8
COLOR 10  NAME=ORANGE     PALETTE=(63,31, 0)  LEVEL= 8
COLOR 11  NAME=PURPLE     PALETTE=(21, 0,14)  LEVEL= 3
COLOR 12  NAME=DIM_RED    PALETTE=(22, 0, 0)  LEVEL= 3
COLOR 13  NAME=DIM_BLUE   PALETTE=( 0, 0,22)  LEVEL= 3
COLOR 14  NAME=DIM_GREEN  PALETTE=( 0,22, 0)  LEVEL= 3
COLOR 15  NAME=HI        PALETTE=(63,63,63)  LEVEL=15

COLOR BLACK  ONE_DOT=WHITE  FOUR_DOTS=(WHITE, WHITE, WHITE, WHITE)
COLOR WHITE  ONE_DOT=BLACK  FOUR_DOTS=(BLACK, BLACK, BLACK, BLACK)
COLOR YELLOW ONE_DOT=YELLOW FOUR_DOTS=(YELLOW, YELLOW, YELLOW, YELLOW)
COLOR GREEN  ONE_DOT=GREEN  FOUR_DOTS=(GREEN, GREEN, GREEN, GREEN)
COLOR RED    ONE_DOT=RED    FOUR_DOTS=(RED, RED, RED, RED)
COLOR CYAN   ONE_DOT=CYAN   FOUR_DOTS=(CYAN, CYAN, CYAN, CYAN)
COLOR BLUE   ONE_DOT=BLUE   FOUR_DOTS=(BLUE, BLUE, BLUE, BLUE)
COLOR MAGENTA ONE_DOT=MAGENTA FOUR_DOTS=(MAGENTA,MAGENTA,MAGENTA,MAGENTA)  &
COLOR GRAY   ONE_DOT=BLACK  FOUR_DOTS=(BLACK, WHITE, WHITE, WHITE)
COLOR BROWN  ONE_DOT=RED    FOUR_DOTS=(GREEN, RED, RED, YELLOW)
COLOR ORANGE ONE_DOT=RED    FOUR_DOTS=(RED, YELLOW, YELLOW, RED)

```

(Continued on next page.)

```

COLOR PURPLE      ONE_DOT=MAGENTA  FOUR_DOTS=(BLUE, MAGENTA,MAGENTA,BLUE)
COLOR DIM_RED     ONE_DOT=RED      FOUR_DOTS=(RED,  WHITE, WHITE, WHITE)
COLOR DIM_BLUE    ONE_DOT=BLUE     FOUR_DOTS=(BLUE, WHITE, WHITE, WHITE)
COLOR DIM_GREEN   ONE_DOT=GREEN    FOUR_DOTS=(GREEN, WHITE, WHITE, WHITE)
COLOR HI          ONE_DOT=BLACK    FOUR_DOTS=(BLACK, BLACK, BLACK, BLACK)

GRID 1  ON  COLOR=RED  DOTS  STEP=1.0
GRID 2  ON  COLOR=CYAN  CROSSES  STEP=5
GRID 3  OFF  COLOR=WHITE  LINES  STEP=50000

LAYER *  WIDTH=2.0  SPACE=0.0  YELLOW PAT=0  NO_PEN
INITIALIZE LAYERS 0:255
LAYER 0 PEN=0
LAYER 1 NAME=NWEL  WIDTH=3.000  SPACE=0.0  BLUE  PAT=0  PEN=16  NO_CIF  NO_STREAM
LAYER 2 NAME=NDIF  WIDTH=3.000  SPACE=0.0  GREEN  PAT=1  PEN=*  NO_CIF  NO_STREAM
LAYER 3 NAME=PDIF  WIDTH=3.000  SPACE=0.0  YELLOW PAT=1  PEN=*  NO_CIF  NO_STREAM
LAYER 4 NAME=PSEL  WIDTH=3.000  SPACE=0.0  YELLOW PAT=0  PEN=*  NO_CIF  NO_STREAM
LAYER 5 NAME=POLY  WIDTH=2.000  SPACE=0.0  RED  PAT=1  PEN=*  NO_CIF  NO_STREAM
LAYER 6 NAME=M1  WIDTH=3.000  SPACE=0.0  CYAN  PAT=2  PEN=*  NO_CIF  NO_STREAM
LAYER 7 NAME=M2  WIDTH=4.000  SPACE=0.0  BLUE  PAT=3  PEN=*  NO_CIF  NO_STREAM
LAYER 8 NAME=CONT  WIDTH=2.000  SPACE=0.0  WHITE  PAT=1  PEN=*  NO_CIF  NO_STREAM
LAYER 9 NAME=VIA  WIDTH=3.000  SPACE=0.0  GRAY  PAT=1  PEN=*  NO_CIF  NO_STREAM

GLOBAL KEY.CF9="@UNED"
GLOBAL KEY.F1="RULER"
GLOBAL KEY.F7="DOS"
GLOBAL KEY.F8="@DEEPSHOW"
GLOBAL KEY.F9="DOS"

```

Figure 3: Sample startup file Q:\ICWIN\TECH\SAMPLES\NEW.CMD

Creating a New Startup Command File

There are several types of environment and technology settings that must be initialized with a startup command file. It can be tedious to write a comprehensive file from scratch. Two better methods of creating your startup command file are listed on the next page.

Avoid using blanks in the name of any command file.

In either case, the your new startup command file should be located in an appropriate subdirectory of the Q:\ICWIN\TECH directory. If one does not exist, you should create it. For example, the technology-dependant command files for a bipolar technology (including the startup command file) might be located in a directory similar to Q:\ICWIN\TECH\BIPOL.

You can open an ASCII text editor by typing NOTEPAD in a console window.

- Copy Q:\ICWIN⁷\TECH\SAMPLES\NEW.CMD and then edit the file with any ASCII text editor.
- Open the layout editor to open an existing cell with the appropriate environment and use the TEMPLATE command to export all environment settings in the cell to a new command file. (This how the sample NEW.CMD was created.)

Environment Settings in Startup Command Files

The following cell environment settings should be made in any customized startup command file.

Color Definitions

Refer to the COLOR command on page 167 for more important information on color definitions.

There are default color definitions for all 16 colors in a new cell even when a startup command file is not assigned. However, no colors will have FOUR_DOTS parameters assigned for 16-color plotting. Also, colors 8-15 will have no ONE_DOT color assigned for 8-color plotting. So it is best to assign all 16 colors in the startup command file with COLOR commands.

You can change any of the values used in the sample startup command file. For example, you may change palette values, names, and plot colors. Color 0 is the background color so changes to its palette will change the background of the editor window.

⁷ Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

You can assign the color definitions in multiple COLOR commands as is done in the sample startup command file, or you can combine each color definition into a single line if you prefer. For example the pair of COLOR commands below defines the exact same color properties as the single COLOR command beneath them. The single command is written with the continuation character '&' on two lines to fit on the page. The command could be written on a single line if you prefer.

Example:

```
COLOR 0 NAME=BLACK PALETTE=( 0, 0, 0)
COLOR BLACK ONE_DOT=WHITE FOUR_DOTS=(WHITE, WHITE, WHITE, WHITE)

COLOR 0 NAME=BLACK PALETTE=( 0, 0, 0) ONE_DOT=WHITE      &
      FOUR_DOTS=(WHITE, WHITE, WHITE, WHITE)
```

The COLOR commands should come before the blocks of GRID and LAYER commands since those commands refer to the color names assigned by the COLOR commands.

Grid Definitions

The NDIV command line option determines the finest granularity of all coordinates. See page 76.

You define three very different types of grids with the startup command file. All three types should be assigned in all startup command files. Each type of grid is assigned by a different command.

RESOLUTION Defines the resolution grid that determines the finest granularity of all coordinates digitized with the mouse. This grid is not usually changed once set.

SNAP Assigns a temporary resolution grid that can be more restrictive than the grid set with the RESOLUTION command. This grid may be changed as necessary with additional SNAP commands in the editor as needed. Different cells in a given project can (and often do) have different snap grids.

GRID Defines up to three display grids that are drawn as dots, crosses, or lines on the screen. You usually assign the finest grid (GRID 1) to be the same as the resolution grid, or a small multiple of this grid. The other two grids are courser grids based on grid 1. All three grids turn on or off automatically as you zoom the scale of the display.

The settings in the sample startup command file define a rather coarse grid for the resolution grid. Unless your technology restricts all coordinates to a .5 micron grid, you may want to modify the RESOLUTION command in your copy of the startup command file. The sample file also defines the snap grid to be the same as the resolution grid. You may also want to modify the initial SNAP grid.

Layer Definitions

Refer to the LAYER command on page 216 for more important information on layer definitions.

Layer definitions may be the most important part of a startup command file. No layers are assigned names or any other properties unless the cell is created with a startup command file with appropriate LAYER commands.

The geometry database of a cell stores the layer number of each component. The correspondence between layer numbers and layer names is made with LAYER commands in the startup command file. If a cell is created with one version of a startup command file and the layer numbers are later redefined with different names in a modified startup command file, new cells will be defined with the new names, but the old cell will still use the old names. Re-executing the new startup command file on the old cell may not solve the problem if the geometry was created on the wrong layer numbers.

For reasons like this, we strongly recommend creating the layer definitions in your startup command file with care, so changes are not required later.

The LAYER commands assign the following properties to each layer number:

- Name
- Default wire width
- Default minimum spacing
- Color
- Screen fill pattern number
- Plot fill pattern number
- CIF export layer name
- Stream export layer number, text type, and data type

The CIF and Stream properties are not stored in the cell file. Refer to Export Properties below.

In addition to the LAYER commands that assign properties to each layer number, there are two other layer-related commands that should be included in a startup command file. The LAYER * command assigns layer properties to all unnamed layers. The INITIALIZE LAYERS 0:255 command removes all unique properties from all layers and gives them the properties assigned by the LAYER * command. The combination of these two commands strips all obsolete properties from all layers in an existing cell.

When you copy the sample startup command file, you can change the properties of the LAYER * command, but be sure not to delete it or the INITIALIZE command. Be sure that these commands precede the other LAYER commands.

Export Properties

The sample startup command file does not assign values to the CIF or Stream export properties. This means that you will not be able to export a CIF or Stream file unless you update these properties.

However, even when you do assign the export properties with CIF or STREAM parameters of the LAYER commands in the startup command file, these properties are not saved with the cell file. The risk of exporting a file with obsolete export properties is too great. So you must assign these properties in the same layout editor session as the CIF or STREAM command used to export a design. The easiest method is to assign these properties in your startup command file, then execute the current version of the startup command file just before the export command. You can use the @* command to execute the current startup command file in the editor.

Another method is to make these assignments in a separate command file that you execute before the export command.

Function Key Assignments

The lines beginning with "GLOBAL KEY." are function key assignments. They assign command strings to function keys. The assigned command is executed when the corresponding key is pressed. These key assignments are often referred to as key shortcuts.

If you make changes to the function key assignments, you may prefer to write these lines in KEY command syntax. (Refer to the KEY command on page 213.) The lines in the sample startup command file use the general macro definition syntax described in the Command File Programmers Reference Manual.

Example: **GLOBAL KEY.F1="RULER"**

This line in the sample startup command file results in the same key assignment as the following KEY command. The RULER command will be executed when the <F1> key is pressed. The KEY command is just a simpler way to write function key assignments.

Example: **KEY F1="RULER"**

Key assignments are one of the most common customizations done by startup command file writers. Feel free to create as many function key assignments as you desire.

Other Commands Common in Startup Command Files

The beginning of the sample startup command file is a collection of miscellaneous environment definition commands. You will need to refer to the individual command descriptions to learn more about these commands if you wish to change a value. We recommend leaving all settings as shown unless you have a need to change one.

VIEW OFF	Prevents update of display while command files execute.
PATTERN "SAMPLE"	Loads the fill patterns stored in the auxiliary file SAMPLE.DAT.
FILL MIXED ON	Turns on display of fill patterns.
AUTOPAN ON PIXELS=100 SECONDS=0.5	Turns autopan feature on so that the view window follows the cursor.

ARROW MODE=EDIT Sets default mode for arrow keys to manipulate command history.

DISPLAY CELL_LABELS=ON OUTLINE_DEPTH=1 EDIT_STACK=OFF
 Sets various display parameters.

CURSOR 1 SNAP=ON Sets cursor style.

SPACER OFF SPACE=0.0 TRACK_LAYERS=OFF STYLE=1
 Initializes then disables display of the wiring spacing cursor.

VIEW LIMIT ON SCALE=0.500 DEPTH=1 DOTS=0 UNITS=0.0 SHOW_LAYERS 1:100
 Controls how data is displayed at coarse scales.

ARRAY DRAW_MODE=SIDES CELL_LIMIT=1024
 Controls display of arrays.

TEXT LOWER_CASE=DISABLED MULTI_LINE =DISABLED ORIENTATIONS=2 &
DISPLAY_ORIGINS=OFF
 Defines display of text components

USE TEXT_JUSTIFICATION=LB WIRE_TYPE=2 ARC_TYPE=2 N_SIDES=16
 Sets defaults for ADD commands.

NEAR UNITS=0.05 DOTS=4 Defines size of near box for selection cursor.

Comments

Refer to the <i>\$comment</i> command on page 120 to learn more about comments.	The line(s) beginning with a \$ in the sample startup command file are only comments. They have no effect on the environment of a cell.
	You can add comments of your own with \$ comments or ! comments. \$ comments will be recorded in the journal file when the startup command file is executed. ! comments are ignored entirely by the command parser.

Adding a Success Comment

The final command in a startup command file will be reflected on the history line of the editor window when the layout editor opens. It is a good programming practice to add the following line as the **last line** of your startup command file:

Example: **\$ %START.CMD executed successfully**

The command line above will leave a comment on the history line (and in the journal file) of exactly which startup command file was executed. The string "%START.CMD" refers to a system macro that stores the entire name (including directory path) of the startup command file. This line does not need to be modified in your various copies of the startup command file. It will always report the correct full name of the file.

Users have been confused in the past about exactly which copy of their startup command file was executed when they changed their command file search path. Adding this line makes it obvious that the correct startup command file was executed.

Always Command Files

See details on the ALWAYS command line option on page 58.

The ALWAYS option on the ICED.EXE command line is used to always execute a command file before opening the editor. You may want to execute your startup command file every edit session rather than only for sessions where you are creating new cells.

Alternately, you can combine the STARTUP and ALWAYS command line options to define both a startup command file and an always command file. This allows you to reserve the startup command file to initialize technology settings in new cells, while using the always command for definitions or overrides that should be made every edit session.

Possible uses of always command files are:

- define macros that are not saved in the cell file,
- save information about the cell in tracking files,
- execute external programs or open console windows, and
- override user settings (e.g. DISPLAY or KEY assignments) in shared cells.

Hierarchical Startup Command Files

You can divide a startup command file into several command files and call these command files from your customized startup command file. This allows some of your settings to be used by several different technologies. It also simplifies the startup command file.

The
`@file_name`
command
executes a
command file.

For example you can modify the color definitions in the file `Q:\ICWIN\AUXIL\COLORS.CMD`. Then call this file from your various startup command files with the command `@Q:\ICWIN\AUXIL\COLORS.CMD` rather than include all of the COLOR commands in each startup command file. In this case, all of your various technologies will use the same color definitions.

Changing a Startup Command File

You should take the time to carefully create the startup command file for your technology before creating any cells. **If you make changes to your startup command file, only new cells will use the new settings.** All existing cells will still use the obsolete settings from the previous version of your startup command file.

Discrepancies between old cells and new cells caused by changes to colors, plot patterns, or layer names in a modified startup command file should not cause significant problems. However, if you change the layer name-layer number correspondences, and you already have existing cells, discrepancies can be disastrous. (See below.)

Executing a Modified Startup Command File

See page 116
for details on
the `@*`
command.

You can use the `@*` command in an existing cell to execute a modified startup command file and load the new settings. (See the AllCells utility on page 378 to execute `@*` in all cells of a library. See the "Looping Through All Subcells"

tutorial in the Classroom Tutorials Manual to see a method of executing `@*` in all cells nested in your main design cell.)

Alternatively, you can have the current startup command file executed every time you launch ICED™ to edit any cell by using the ALWAYS command line option instead of STARTUP on the ICED.EXE command line.

Changing Layer Numbers in Existing Cells

Changes in layer name-layer number correspondences in the startup command file must be handled with extreme care once some cells in the design are already created.

For example, let us say that the following line was in the startup command file used for some cells in your design:

LAYER 10 M1

This setting is saved in the environment database of all of those cells. When you edited any of those cells and specified the M1 layer when adding a component, the component is stored on layer number 10. The layer number of all components is stored in the geometric database of the cell.

Now say you change this line in the startup command file to the following:

LAYER 11 M1

Now when you create a new cell, this is the setting stored in the environment database of the new cell. When you add a component on layer M1 in these cells, it is stored on layer number 11.

The change to the startup command file has not changed either the geometry or environment database of the old cells. Components that you consider to be on layer M1 are now on two different layer numbers in various cells.

Executing the modified startup command file on the old cells will not solve the problem. Only the environment settings in the cell will change. The components are still on layer number 10 in the geometry databases of the old

cells. You must change the layer number of components and then execute the new startup command file in all of the old cells.

If you must change layer name-layer number correspondences in the middle of a design, there are a few different ways of doing this. The method you choose depends on what cells need to be modified.

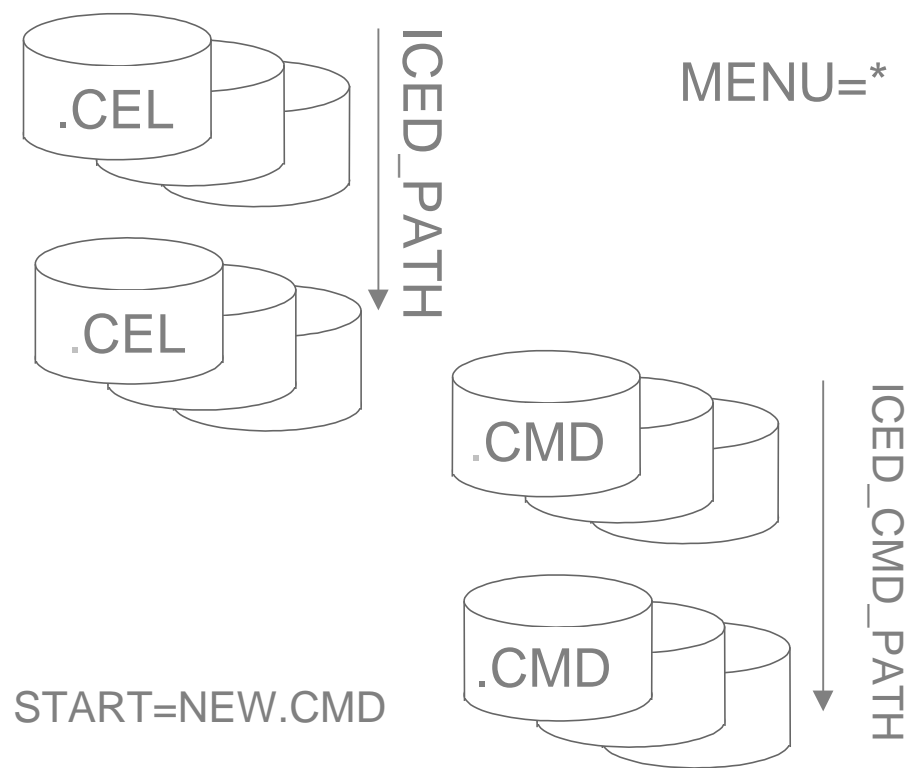
If you need to modify all cells in a library then use the **AllCells utility** to create a batch file that will open each cell in a library and execute a given command file. See the example on page 378. This is the slowest method, but it is the only method that will guarantee that all cells in a library are changed even when they are not nested in a design cell. This also the only method that will leave cells in their original libraries

If you need to change only a few cells, you can open the layout editor and edit each cell using an **EDIT CELL command**. You may want to create a command file that contains the commands to select and swap the layers of the components to be sure you perform the same procedure in each cell. (See the example command file on page 380) In this case modified cells from protected libraries will be stored in the working directory.

If you need to change only subcells of your main design cell, you can use the **_LOOP.CMD** or **_LOOP2.CMD** command files in the layout editor to execute commands (or a command file) on all subcells of your main cell. This will also save modified cells from protected libraries in the working directory. (See the descriptions of these command files in the Classroom Tutorials Manual.)

One last method to change the layer numbers of all cells nested in your design is to export your entire design with the **STREAM ARCHIVE command**, and then import it again with the **UnStream utility**. During the UnStream process, specify an alias file that translates the layer numbers of existing components appropriately. This process strips the environment from all cells so that the new startup command file is executed automatically the first time each cell is opened with the layout editor. All cells will be recreated in the working directory so you will need to move cell files that should be located in separate cell libraries.

Project Batch Files



See an explanation of methods for executing project batch files on page 90.

The usual method for opening the ICED™ layout editor is to execute a batch file from the console window opened with the ICED icon on your desktop. We call this batch file a **project batch file** because it is customized with settings specific to a project, e.g. the cell library search path and the startup command file. The simplest project batch file contains the following:

- SET commands to define search paths stored in system environment variables, and
- the ICED.EXE command line.

Sample ICWIN.BAT

The sample project batch file supplied with the installation is shown below.

```
CLS
SET ICED_HOME=Q:\ICWIN;
SET ICED_PATH=Q:\ICWIN\SAMPLES;
SET ICED_CMD_PATH=Q:\ICWIN\TECH\SAMPLES;
SET DRC_PATH=Q:\ICWIN\TUTOR;
IF %1==_ GOTO DONE
Q:\ICWIN\ICED %1 START=Q:\ICWIN\TECH\SAMPLES\NEW MENU=M1 PAUSE=0 ...
... MAXIMIZE=YES WIN=500
:DONE
```

Figure 4: Q:\ICWIN\ICWIN.BAT

The environment variables are explained in more detail on page 43.

The line beginning with "IF" is a conditional statement that controls what the batch file will do depending on whether or not an argument is included in the batch file command line.

Value of first argument	What ICWIN.BAT will do
(None)	Create environment variables and quit
<i>cell_name</i>	Create environment variables and open layout editor to edit this cell as the root cell.

Figure 5: Effect of arguments on ICWIN.BAT batch file

Example: **ICWIN MYCELL**

If the layout editor is running, you can open a console window with the environment variables already defined using the DOS or SPAWN commands.

When the command line above is typed at the console prompt, it will open the layout editor to edit the cell MYCELL in the current directory. Note that you do not need to type the .BAT file extension to execute the batch file.

When ICWIN.BAT is executed without arguments, it can be used to only define the environment variables. This can be useful when executing ICED™ utilities outside of the layout editor or in another batch file. The layout editor command line is **not** executed in this case.

Example: **ICWIN
ICTOP**

The ICED™ utilities are described beginning on page 369.

Since no arguments are included in the ICWIN.BAT command line in the example above, the batch file will define the environment variables (including the cell library search path) and then it will quit without opening the layout editor. The second command executes the ICTOP.EXE utility that uses the cell library search path to help locate top-level cells in the current directory.

Creating a New Project Batch File

You can open an ASCII text editor by typing NOTEPAD in a console window.

To create your own project batch file, copy the sample file (Q:\ICWIN\ICWIN.BAT) to a new file with the same file extension in the same directory. Do not use blanks in the file name. Edit as required with any ASCII text editor.

Comments can be added to a batch file by beginning the line with the string "REM" (for REMARK).

It is best to keep all batch files in the Q:\ICWIN directory. This directory is automatically added to the PATH system environment variable (that defines the executable file search path) in the console window opened by the ICED desktop icon.

When you execute a batch file (using a .BAT file extension), the MS-DOS console command interpreter is invoked. This allows you to use commands in your batch file like SET (for setting environment variables) or COPY (for copying files). The batch file can include commands to execute any program. Conditional statements can be added to control the execution of the batch file under different conditions.

You can create a batch file that executes layout editor tasks with no user interaction required. See details on page 90.

If you want to pass other arguments into your batch file you can refer to each argument number preceded by a '%'. Look at the sample batch file on page 40. Note how "%1" appears on the ICED.EXE command line. This string is replaced with the first argument on the batch file command line, the cell name.

If you use the string "%2" in your batch file, it is replaced with the second argument. You can use up to 9 argument placeholders in this manner. See an example on page 53.

Defining Environment Variables

All of the search path environment variables described in the following table should be defined with SET commands in a batch file used to open the layout editor. These definitions must come **before** the ICED.EXE command line. We will cover each variable with examples on the following pages.

Environment variable	Page	Purpose
ICED_HOME	44	Defines ICED™ directory trees that will be searched for a variety of program support files .
ICED_PATH	45	Defines cell library search path.
ICED_CMD_PATH	48	List of directories searched for command files .
PATH	49	System search path for executable files .
DRC_PATH ⁸	50	Search path used for DRC rules files

Figure 6: Description of ICED™ environment variables

Displaying Current Environment Values

You can display the current values of all environment variables with the following MS-DOS command in any open console window.

Example: **SET**

The values of the search paths defined with the following environment variables are also recorded automatically in the launch messages when the layout editor opens. (Refer to the PAUSE command line option on page 77 to see details on viewing these messages.) These messages are also saved in the journal file.

⁸ The DRC_PATH environment variable needs to be defined only when you may be executing the DRC program.

SET ICED_HOME[_n]= *dir1* [;*dir2*] [;*dir3*]

See page 22 for some reasons why you may want to have more than one directory on the auxiliary file search path.

This environment variable defines the home directories ICED™ will search for **support files**, particularly auxiliary files like menu files. When ICED™ searches for auxiliary files, it will look in the AUXIL subdirectory of the first home directory, then in the AUXIL subdirectory of the second home directory, etc.

Normally, the only home directory is the Q:\ICWIN⁹ installation directory. A typical ICED_HOME definition would look like the following example.

Example: **SET ICED_HOME=Q:\ICWIN**

You can define up to three different directory trees separated with semicolons (;). The brackets ([]) in the syntax above (and elsewhere in this manual) indicate that the parameter is optional.

Example: **SET ICED_HOME=Q:\ICWIN;N:\ICDIST**

Searching for support files on networked drives will probably slow the response time for some commands.

The SET command above creates an environment variable that defines two home directories. The order in which they are defined will be the order in which they are searched for files. So files in the Q:\ICWIN directory tree will hide files with the same names in the N:\ICDIST directory tree. Suppose N:\ICDIST is an installation directory on a networked computer while Q:\ICWIN contains customized files on your local computer. If the M1.MEN menu file exists in the AUXIL directories of both home directory trees, a customized Q:\ICWIN\AUXIL\M1.MEN file would be used instead of the distribution copy in N:\ICDIST\AUXIL\M1.MEN. However, if a required support file is not found in the Q:\ICWIN\AUXIL directory, ICED™ will search for the file in the N:\ICDIST\AUXIL directory on the network.

The '&' is an obsolete line continuation character. Use '+' instead.

The definition of ICED_HOME can be split over several lines. If you add the character '+' at the end of the SET ICED_HOME command, ICED™ will concatenate the value of the variable ICED_HOME_1 to ICED_HOME. A '+' at the end of ICED_HOME_1 will add the optional ICED_HOME_2 to ICED_HOME. However, a single directory path cannot be split over two lines.

⁹ Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

Example: **SET ICED_HOME=Q:\ICWIN;+**
 SET ICED_HOME_1=N:\ICDIST

Each of these directories should have the subdirectory structure set up by the installation program.

The maximum number of characters in each path is MAXPATH-50, where MAXPATH is a system dependant value, usually set to 250. Therefore, it is best to keep each path name shorter than 200 characters.

SET ICED_PATH[_n]=*cell_lib_1* [/*class_1*]; [*cell_lib_2* [/*class_2*];]...

The []'s
indicate
optional
parameters.

The environment variable ICED_PATH defines the list of directories ICED™ will search for **cell files**. ICED™ will search for cells in each of the directories in the order they are defined. A cell in the first cell library will hide a cell with the same name in any of the other libraries.

Each optional library protection *class_n* parameter associates a library protection class with the corresponding cell library. The possible values are shown in Figure 7. The class determines whether or not cells in the library may be modified, and where modified cells will be stored.

<i>/class_n</i> switch		Class name	Purpose
/D		Direct Edit	Cells in libraries of this class can be modified and then saved in the same library without restrictions.
P r o t e c t e d	/C	Copy Edit	Cells in libraries of this class can be modified, but when they are saved, they will be saved in the working directory rather than in their original library. The library itself remains intact.
	(default) /V	View Only	Cells in view-only libraries cannot be modified unless they are selected as the working directory when you open the layout editor.

Figure 7: ICED_PATH cell library classes

The direct-edit class is the only unprotected cell library class. Note that the direct-edit class is not the default class. In fact when no switch is used, **cell libraries default to the view-only class**. This is the case because you can overwrite cells unintentionally when a cell library is not protected. The working directory is always a direct-edit cell library, and it is often the only unprotected cell library. You intend to change cells in the working directory. However, you do not want to edit a subcell used elsewhere in the design, or even in other projects, without careful consideration.

Libraries not marked with a direct edit or copy edit protection class switch are automatically view-only libraries.

Example: **SET ICED_PATH=C:\ICED\SAMPLES/C; C:\ICED\CMOS**

Note that libraries are separated with semicolons (;). Blanks can be used after the semicolons for readability. They will be ignored.

In the example above, the C:\ICED\SAMPLES cell library will be defined as a copy edit cell library. C:\ICED\CMOS will be defined as a view-only library.

The ICED™ subcell edit commands are **EDIT**, **P_EDIT**, and **T_EDIT**.

Let us say that you need to edit a subcell in the C:\ICED\SAMPLES copy-edit library, and that library is NOT selected as your working directory. When you use a subcell edit command (EDIT, P_EDIT or T_EDIT) to edit the cell in the copy-edit library, ICED™ will warn you that the cell is in a different library. You will be provided with options to only view the cell (you will not be able to save any changes you make to the cell), to edit a local copy of the cell (which will be saved to your working directory), or to cancel the edit. In no case will cell files in the copy-edit library be overwritten. This prevents changes to any cell file stored in a copy-edit library, but allows you to create a local copy of the cell for your own use.

You can prevent changes to any cell by using the **VIEW_ONLY=ON** command line parameter.

The cells contained in the C:\ICED\CMOS cell library directory cannot be edited. (The only exception to this rule is when ICED™ is launched to edit a root cell in that directory. See below.) If you try to use a subcell edit command to edit a cell in a view-only library, ICED™ will warn you that the cell is contained in a view-only library and provide you with only the view-only or cancel options.

See more details on working directories on page 19.

When you launch ICED™, you usually first use a CD (change directory) command to navigate to the directory where the cell file you wish to edit is stored. This cell is referred to as the **root cell**. The directory is referred to as the **working directory**. The working directory is always marked as a direct edit library. Any cell contained in a direct edit library can be edited and overwritten without warning. **ICED™ always searches for cells first in the working directory.** Cells in the working directory will hide cells with the same names in any other cell library.

Example:

SET ICED_PATH=C:\BIPOLAR\CURRENT/D;

The "/D" switch in this command will mark the C:\BIPOLAR\CURRENT directory as a direct edit library. Cells contained in this directory can be edited and overwritten without warning.

The '&' is an obsolete line continuation character. Use '+' instead.

You can spread the ICED_PATH definition over more than one line using the same + syntax as the ICED_HOME definition. For example, adding the character '+' at the end of the SET ICED_PATH command will make ICED™ concatenate the value of the variable ICED_PATH_1 to ICED_PATH. A single cell library directory path cannot be split over two lines.

Example:

**SET ICED_PATH=C:\BIPOLAR\CURRENT/D; +
SET ICED_PATH_1= E:\BIPOLAR\SHARED/C; +
SET ICED_PATH_2= E:\BIPOLAR\LIBRARY**

See several scenarios on cell library definition with shared cell libraries in the Classroom Tutorials Manual.

ICED™ will search for cell files first in the working directory, then in each of the libraries specified in the ICED_PATH parameter, in order. If you have two cell files with the same name, (perhaps one is a local copy from a copy edit library) one copy or the other may be found first in a later ICED™ session. It will depend on the directory where the current root cell is stored, and the order of the libraries in the ICED_PATH parameter in effect.

To learn more about the launch messages, refer to the PAUSE command line option on page 77. The journal file is described on page 99.

The cell libraries defined in ICED_PATH are reported in the launch messages shown in the console window when the layout editor opens. These messages are saved in the journal file as well.

The maximum number of characters in each directory name for this environment variable (and the next) is MAXPATH-40, where MAXPATH is a system dependant value, usually set to 250. Therefore, it is best to use directory names of 210 characters or less.

The maximum number of cell libraries you can define is 250.

SET ICED_CMD_PATH[_n] = *cmd_file_dir_1*; [*cmd_file_dir_2*;]...

The []'s indicate optional parameters.

The ICED_CMD_PATH environment variable controls the search path for **command files**. You can always include the path to the command file in the command that calls the command file. However, when you do not supply this path, ICED™ must search for the command file. This search path is built automatically by the layout editor as follows:

Learn more about command files on page 14.

- the current working directory
- directories in the ICED_CMD_PATH list, and finally
- the AUXIL subdirectory(ies)

See page 44 to learn about using multiple AUXIL directories.

A given command file in the working directory will hide any other command files with the same name in the other directories. Command files are looked for in the AUXIL directory(ies) only when they are not found in the other directories. This is the usual location for command files that are technology-independent.

Example:

SET ICED_CMD_PATH=E:\BIPOLAR

With this simple ICED_CMD_PATH definition, the search order for command files will be: the current working directory, then E:\BIPOLAR, then each of the AUXIL subdirectories of the directories defined in the ICED_HOME variable.

See page 24 for an overview of the TECH directory.

This environment variable is not required if you put all of your command files in the AUXIL subdirectory(ies). However, we strongly recommend placing technology-dependant command files in their own separate directory. For

example, keep all command files that are used for cells to be fabricated with a CMOS .5 micron process in a directory with the name similar to Q:\ICWIN\TECH\CMOS.5. Then add this directory to the command file search path only for projects using this technology with an ICED_CMD_PATH definition similar to the one below.

Example: **SET ICED_CMD_PATH=Q:\ICWIN\TECH\CMOS.5**

When you migrate to a new technology, copy all files to a new directory (e.g. Q:\ICWIN\TECH\CMOS.2) then edit the files for the new technology. In projects using the new technology do not add Q:\ICWIN\TECH\CMOS.5 to the command file search path. Use the new directory instead. This also keeps the old files intact for reference or for projects that use the older technology.

Older versions of ICED™ would search for command files in the cell libraries, but this is no longer the case. If you do want to use the old method of searching cell libraries for command files, you can do this easily by using the ICED_PATH definition in the ICED_CMD_PATH definition.

Example: **SET ICED_CMD_PATH=%ICED_PATH%**
 SET ICED_CMD_PATH_1=%ICED_PATH_1%
 SET ICED_CMD_PATH_2=%ICED_PATH_2%

If the example above comes **after** the lines of the ICED_PATH example on page 47, the same directories will be used for the cell file search path and the command file search path. If you change one of these directories, you need only edit the ICED_PATH definition and the ICED_CMD_PATH definition change is automatic.

SET PATH = exe_file_dir_1; [exe_file_dir_2;]...

The []'s indicate optional parameters.

This is a system environment variable that controls the search path for **executable files** used by the command interpreter. It affects where the operating system will search for all programs, not just ICED™ programs. If you are executing your batch file in an open console window, the change made to your system PATH variable will be in effect even after the batch file is complete.



Figure 8:
ICED
desktop
icon

Usually, you do not need to add this environment variable definition to your batch file(s). The Q:\ICWIN¹⁰ directory is automatically added to the front of the system PATH when you open a console window with the ICED icon on your desktop. Executing your project batch file from this console window is the recommended method for opening the layout editor.

Your operating system probably defines a default PATH environment variable. This allows you to execute programs in a console window without explicitly typing the location of the executable file in front of the program name. It is a good idea to avoid omitting any directories already defined in the PATH variable when you modify it for ICED™ use.

Example: **SET PATH=%ICED_HOME%;%PATH%**

The example above will add the ICED™ home directories (already specified with the ICED_HOME environment variable) to the front of the existing system PATH variable. This means that programs stored in the ICED™ home directories will be found even if you do not type the name of the path to the program. Since these directories come first in the PATH definition, even if programs with the same name exist in other directories on the PATH, the ICED™ programs will be found first by the operating system.

SET DRC_PATH = *drc_file_dir_1*; [*drc_file_dir_2*;]...

The []'s indicate optional parameters.

The Classroom Tutorial Manual to learn more about the internal DRC options.

This environment variable controls the directories searched for **DRC (Design Rules Checker) support files**. The DRC command in the layout editor may require this variable to find a rules file. The internal DRC options available from the DRC menu also use this option to search for rules files.

¹⁰ Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

ICED.EXE Command Line Syntax

See page 39 for an overview of project batch files.

Type ICED.EXE without arguments to report version.

The usual method to open the ICED™ layout editor is to create a project batch file that includes the ICED.EXE command line. You then execute the batch file from a console window (also known as a MS-DOS prompt window) opened by the ICED icon on your desktop,

Create the command line with the appropriate arguments for ICED.EXE program using the syntax covered below.

[*prog_path*\] ICED [*cell_path*/*cell_name* [*parm1*=value]...[*parmn*=value]

The items in []'s are optional. Do not use the []'s when typing the command line. The only required parameter is *cell_name*. This is the name of the root cell you wish to edit. It can be the name of an existing cell or a new cell you wish to create.

Example: **Q:\ICWIN\ICED CELLX**

The optional *prog_path* parameter defines the drive and directory location of the ICED.EXE program. In the example above, *prog_path* is "Q:\ICWIN". The *prog_path* parameter is not required if the directory where the program is located is included in your PATH environment variable. (See page 49.)

See page 96 for restrictions on the names of cells.

In the example above, *cell_name* is "CELLX". The extension of the cell file (i.e. .CEL) is not required. The editor will open and the cell file CELLX.CEL will be loaded **if it exists in the current working directory**. The cell libraries defined by the ICED_PATH environment variable are **not** searched for the root cell.

If the cell file is (or will be) located in a different directory, use the *cell_path* parameter to specify the drive and path to the cell directory. If the cell file is (or will be) stored in the current directory, *cell_path* is not required.

See an example of navigating to the working directory with a MS-DOS CD command on page 23.

You usually use the MS-DOS CD command to make the current directory the working directory where your cell files are (or will be) stored, prior to executing the command line. In this case you do not need to supply *cell_path* in the command line.

When you do supply *cell_path* in the command line, *cell_path* becomes the working directory. This is the directory where the journal file will be stored as well as any new cell files. (If this is inconvenient, use the nested edit method described on page 20 and specify a dummy cell in a convenient working directory on the command line.)

Example:

Q:\ICWIN\ICED Q:\BIPOLAR\CELLS\CELLX

Refer to page 19 to learn more about working directories.

In this example, Q:\BIPOLAR\CELLS will be used as the working directory. The journal file and all new cells files will be stored there. The cell file Q:\BIPOLAR\CELLS\CELLX.CEL will be loaded into the editor if it exists. If not, this file will be created when the editor closes.

Any optional parameters must be typed after *cell_name*. You do not need to type the other parameters in any specific order. These parameters control the layout editor settings. For example, the MENU parameter determines the command menu used and the STARTUP parameter defines the startup command file used to initialize technology settings. Each parameter is covered in alphabetical order in Command Line Parameters beginning on page 54.

The first few characters of the parameter keyword is usually sufficient to specify it. Parameters are processed from left to right. If you use a parameter twice, or use mutually exclusive parameters, only the last parameter processed will be used. (The only exception to this rule is the *@file_name* parameter used to load command line options from a file.)

Example:

ICED CELLX MAX=YES

Since no *prog_path* is provided in this command line, ICED.EXE must exist in the current directory or in a directory listed in the environment variable PATH. If the file CELLX.CEL does not exist in the current directory, ICED™ will launch to create a new cell, but since no STARTUP parameter is used, the editor will open without a menu, layer or technology settings. If the file CELLX.CEL

already exists in the current directory, it will be loaded into ICED™ along with all cell environment variables stored in it (e.g. menu and technology settings).

The MAX=YES parameter is a short form of the MAXIMIZE command line parameter. It forces ICED™ to create the editor window in the maximized mode. It will fill your entire screen.

Example:

ICED %1 START=NEW.CMD MENU=M1

The START option defines the startup command file executed automatically for new cells.

The MENU option loads a menu.

When the command line above is used in a batch file, the "%1" placeholder is replaced with the first argument on the batch file command line, the root cell name.

Assume that the command line above is typed in a project batch file with the name MYPROJ.BAT. If you execute this batch file with the following command line, the first argument "MYCELL" replaces the "%1" placeholder everywhere in the batch file.

MYPROJ MYCELL

The ICED.EXE command line that is executed in this case is:

ICED MYCELL START=NEW.CMD MENU=M1

If you want to pass additional arguments when you execute the batch file, you can use %2, %3, etc. in the ICED command line in the batch file. In this case, you may find that parameters of the form *keyword=value* are not passed correctly to ICED.EXE. The '=' may be replaced with a blank space. See the BATCH command line option on page 60 to solve this type of problem.

Command Line Parameters

The first parameter on the ICED™ command line is always the cell name. (See page 51 for details.)

See the example on page 53 to learn how to pass parameters on the project batch file command line to the ICED.EXE command line. The options listed on the following pages can all be used on the ICED™ command line after the cell name.

Some operating systems have a 128-character limit for command lines.

If you use several command line parameters you may want to organize your parameters in an options file rather than type them all on the ICED.EXE command line in your project batch file. In this case, use the *@file_name* option (see page 56) to store parameters in a file rather than include them all on the command line.

(If you do not want to read every command line parameter description now, you can skip ahead to page 87 to learn about memory management, or to page 90 to learn about methods of executing batch files and batch files that open the layout editor to perform tasks without user interaction.)

Command line parameter keyword	Commonly used in project batch files	Page	Purpose
<i>@file_name</i>	Rarely	56	Read command line options from a file
ALWAYS	Rarely	58	Execute command file as soon as editor opens (Similar to STARTUP, but always executed)
BATCH	Rarely	60	Force command line parser into batch mode (Useful when passing parameters on batch cmd line.)
BEEP	Rarely	62	Set warning beep
Continued on next page			

CANCEL	Rarely	63	Assign Cancel to function key (Useful primarily on portable computers when pressing both buttons fails to cancel commands)
D16	Rarely	64	Define units for ICED-16 cell
EXIT	Rarely	65	Execute command file, save cell file, and close editor
ICED	Rarely	67 88	Alternative memory management for huge databases.
LEAVE	Rarely	69	Execute command file and close editor with LEAVE command
LIMIT	Rarely	70	Limits display complexity and speeds up drawing the initial view window (Useful primarily when opening top-level cell)
LOG	Rarely	72	Control journaling
MAXIMIZE	Often	73	Control mode of editor window
MENU	Frequently	74	Override menu
NDIV	Rarely	76	Set divisions per user unit
PAUSE	Often	77	Control how long editor opening messages are displayed
QDIV	Rarely	78	Similar to NDIV, but warning message is suppressed
QUIT	Rarely	79	Execute command file and close editor without saving cell
RAM	Rarely	80 88	Refine memory management for ICED parameter is used
STARTUP	Almost always	82	Execute command file for new cells (Specifies startup command file)
TIME_OUT	Rarely	83	Allow more time for security check
TMP	Rarely	84	Specify directory for temporary files. (Useful primarily for network installations)
VIEW_ONLY	Rarely	85	Prevent changes to cells
WINDOWS	Frequently	86 87	Limit memory available

Figure 9: Summary of ICED.EXE command line parameters

@file_name

Read options from a file

ICED™ has a large number of command line parameters and the command line can grow unreasonably long. To allow for more readable batch files, to reduce repetitive typing, and to avoid command line parsing problems, ICED™ can read command line parameters from an options file.

(On some operating systems the command line parser will not allow more than 128 characters on a command line. The best way to launch ICED™ with more than 128 characters of parameters (including blanks) is to use the *@file_name* command line parameter with a reference to an options file. The total number of characters in the options file can exceed 128-characters.)

Example: **ICED CELLX @Q:\ICWIN\ICEDOPT.TXT**

This command line tells ICED™ to use the command line options found in the file Q:\ICWIN\ICEDOPT.TXT.

The options file can be created with any ASCII text editor. There are no restrictions on the file name. (The only restriction is that you must use quotes around the file name in the *@file_name* command if the file name contains blanks.) You should type one command line parameter on each line. When ICED™ reads an options file, it considers anything to the right of an exclamation point as a comment. Thus both of the lines below mean the same thing to ICED™:

```
QUIT=STREAM_OUT.CMD
QUIT=STREAM_OUT.CMD !Execute stream export command file
```

An options file can have as many lines as are required. (ICED™ ignores blank lines and lines which start with an exclamation point.) A single parameter cannot be split between two lines.

The options file can **not** have another *@file_name* parameter nested in it. Only other ICED™ command line parameters, comments, and blank lines may be included in this file. **The cell name can not be included in an options file.**

The options file is very useful when running ICED™ over a network, since command line parameters that use qualified file names are often used and the directory names on the network can often be lengthy. The options file used when ICED™ is run over a network might look similar to:

Example: **TMP=E:\TMP**
 START=X:\DEPT52SVR\PUBLIC\ICED\TECH\CMOS.CMD
 MAX=YES
 MENU=M1

Once you have the options file created, you can use the *@file_name* parameter on the ICED™ command line. You should include the directory path with the file name unless the options file is located in the working directory. Always include the file extension.

More than one *@file_name* parameter can be used on a single command line. Additional parameters can be typed on the command line as well.

Example: **ICED CELLX @C:\ICWIN\OPT.TXT @C:\ICWIN\SOPT.TXT PAUSE=30**

This command line will launch ICED™ using the command line parameters in C:\ICWIN\OPT.TXT, then the parameters in C:\ICWIN\SOPT.TXT, and finally the PAUSE=30 parameter. The PAUSE=30 parameter will override any PAUSE parameters in the options files and leave the opening messages on your screen long enough for you to see if there were any problems.

ALWAYS=*file_name*

Always execute command file

See an overview on the uses of always command files on page 35.

New with version 4.82 is the ability to combine the STARTUP and the ALWAYS options.

Learn about the many uses of command files in the Command File Programmer's Reference Manual.

Example:

The *@file_name* command can be used to execute any command file once the editor is open.

See page 48 for details on the command file search path.

The ALWAYS parameter causes the ICED™ commands in *file_name* to be executed as soon as you open the layout editor.

This parameter is similar to the STARTUP=*file_name* command line parameter, except that with this parameter *file_name* is executed every time you launch ICED™, not just when you create a new cell. You can use the STARTUP command line option in addition to the ALWAYS option. This allows you to reserve the startup command file for technology settings that should be assigned only in new cell files while allowing you to use the always command file for purposes like overriding display settings, defining key assignments, or defining special purpose macros every edit session.

If no extension is given in *file_name*, .CMD is assumed. If *file_name* is not stored in a directory on the command file search path, you must supply the directory path to the file in *file_name*. If you indicate a file that does not exist, ICED™ will warn you with an error message and an audible beep, but it will not abort.

ICED MYCELL ALWAYS=NEWKEYS.CMD

Using this parameter on the ICED™ command line will result in the commands found in the file NEWKEYS.CMD being executed as soon as ICED™ is launched to edit any cell. This command file may contain KEY commands to create keyboard shortcuts that you continue to update as you work in different cells. Using this option will automatically create these up-to-date shortcuts in every cell you edit. However, if you create a new cell with this command line, the startup command file (that creates the correct editor settings for a new cell) will not be automatically executed.

Since no path is supplied in the example above, ICED™ must be able to find the command file on the command file search path.

If you prefer to have ICED execute a command file and then terminate without any user interaction, use a QUIT, EXIT, or LEAVE command line option instead

of ALWAYS. While these options are not mutually exclusive, the option furthest to the right on the command line will override any others of this group. A warning is posted, but the editor will continue to open after the warning and the single command file will be executed.

If you want the command file executed only for new cells, use STARTUP instead of ALWAYS. When you use both options on the same command line, the startup command file is executed first regardless of the order of the options. If no errors are encountered in the startup command file, then the always command file is executed, then the editor layout window opens.

BATCH

Force command line parser into batch mode

The use of this parameter in the command line will cause two changes:

- 1) Command line parameters to the left of the BATCH parameter can use '#' in place of '='.
- 2) Most launch message warnings will not halt the editor before it goes into graphics mode.

In a batch file, '%1' is replaced with the first argument on the command line when you execute the batch file. '%2' is replaced with the second argument, etc.

Change number 1 is very useful when passing command line arguments on the project batch file command line because the COMMAND.EXE command interpreter (used to parse batch file arguments) will interpret each '=' in a batch file argument as a whitespace character. (That means it replaces all '=' with blanks.) This makes it impossible to pass arguments with the format *keyword=value* into a batch file. Since this is the format of most ICED™ command line parameters, the BATCH parameter is required when you are passing ICED™ command line parameters as arguments into a batch file.

For example, let us assume that you have a project batch file with the name MYPROJ.BAT. You want to use this batch file to launch the layout editor. Assume that the project batch file contains an ICED.EXE command line similar to the following:

Example: **ICED %1 MENU=M1 BATCH %2 %3 %4 %5 %6 %7 %8 %9**

If you call this batch file with the following command that uses two arguments, it will pass both arguments correctly:

Example: **MYPROJ MYCELL QUIT#PLOTALL.CMD**

See another example using this option on page 53.

The command line executed will be equivalent to the following:

ICED MYCELL MENU=M1 QUIT=PLOTALL.CMD

If you called the batch file using the following syntax instead:

Example:

MYPROJ MYCELL QUIT=PLOTALL.CMD

The QUIT command line option executes the indicated command file as soon as the editor opens, then closes the editor automatically without saving the cell.

then the command line executed would be:

ICED MYCELL MENU=M1 QUIT PLOTALL.CMD

which will cause a syntax error since the '=' has been replaced by a blank space.

If the BATCH keyword was not used in the ICED.EXE command line in the batch file, and you used the *keyword#value* syntax for the QUIT parameter (as shown on the previous page), then you would also get a syntax error.

The use of the BATCH parameter must be combined with the *keyword#value* syntax to pass ICED™ command line parameters as arguments into a batch file.

Change number 2 on the previous page (suppression of most launch message warnings) will prevent the layout editor from going into an infinite pause before launching the editor when problems like a missing menu are encountered. This allows your batch file to complete without user interaction even when small problems are encountered.

However, be advised that these warnings will flash by too quickly for you to see when the batch file executes. You can always look at the beginning of the journal file to see these warnings if you get unexpected results.

BEEP=(ON | OFF)

Set warning beep

This parameter enables or disables the error beep. The default is BEEP=ON.

The beep is used by ICED™ to warn you when errors or warnings occur.

Example: **ICED MYCELL BEEP=OFF**

Using this parameter will disable the error beep. This would allow the passenger next to you sleep as you edit cells on your laptop on that airline flight to Asia.

CANCEL=Fn

Assign Cancel Current Command to Function Key

Assigning ordinary command strings to function keys is done with the **KEY** command.

The usual way to cancel a command is to press both mouse buttons simultaneously. The <Esc> key can also be used to cancel a command, but only when coordinates are not being digitized.

Some portable computers (e.g. SONY computers) do not process the pressing of both mouse buttons simultaneously in the usual manner. So ICED™ now allows you to cancel the current command by pressing a function key. You must add a CANCEL=Fn option to the command line to use this feature, where *n* is the number of a function key in the range 1:12.

Example: **ICED MYCELL CANCEL=F9**

This layout editor command line assigns the “Cancel current command” operation to the <F9> function key. In the layout editor session, you will be able to cancel any command by pressing the <F9> key.

D16=*dbu_int*

Define units for ICED-16 cell

This parameter is useful only when you edit a cell created with ICED-16, an older version of ICED™. Both ICED-16 and ICED™ store coordinates in database units. However, the size of the ICED-16 and ICED™ database units are usually different. By default, ICED™ automatically computes the number of ICED™ database units in one ICED-16 database unit. This parameter allows you to specify the ratio manually.

See the **NDIV** command line parameter on page 76 for an explanation of user units and database units.

If you do not specify D16, ICED™ will assume that the ICED-16 user unit is the same size as the ICED™ user unit. If you are working with NDIV=1000 and read an ICED-16 cell that was created with DIV=4, ICED™ will assume that one ICED-16 database unit is equivalent to 250 ICED™ database units.

The D16 parameter is useful if the ICED™ and ICED-16 user units are not the same size. If you specify D16=*n*, ICED™ will assume that one ICED-16 database unit is equivalent to *n* ICED™ database units. This ratio is assumed to hold for all ICED-16 cells you edit.

The *dbu_int* value must be an integer in the range 1:25400.

EXIT=*file_name*

Execute command file and exit

Use the QUIT command line option instead of the EXIT option if you want to avoid saving the cell. Use the LEAVE option instead to save only modified cells.

This option is very similar to the ALWAYS command line option. It executes a command file as soon as the layout editor opens. Unlike the ALWAYS option, the layout editor will close automatically after the command file is finished. If the command file has no errors, the editor will close automatically with an EXIT command that saves the cell file.

If no extension is given in *file_name*, .CMD is assumed. If *file_name* is not stored in a directory on the command file search path, you must supply the directory path to the file in *file_name*. If you indicate a file that does not exist, ICED™ will warn you with an error message in the journal file and then close the editor without saving the cell.

Example:

ICED MYCELL EXIT=SWAPIT

See page 48 for details on the command file search path.

This command line will open the cell MYCELL and execute the commands in the command file SWAPIT.CMD. Since no directory path is included with the command file name, SWAPIT.CMD must exist in the working directory or in one of the directories in the command file search path.

If SWAPIT.CMD contains no errors, the cell file for MYCELL will be saved with an EXIT command and the editor will close.

See page 99 for information on the journal file.

If SWAPIT.CMD does contain an error, then the remaining commands in the file will not be executed and the editor will close with a JOURNAL command. The cell file will not be saved, however the work done by the successful part of the command file can be recovered with the journal file. However, the best method is to correct the error in the command file and repeat the command file from the beginning. Look for the error message at the end of the journal file, correct the command file, delete or rename the journal file, and finally re-execute the ICED command. If the journal file still exists when the ICED command line is re-executed, the user will need to respond to the automatic recovery option by choosing the "NO" option in the automatic recovery dialog box to proceed.

You can create new cells with this option. Include a STARTUP option on the command line to initialize the environment of the new cell.

Example: **ICED NEWCELL STARTUP=NEW EXIT=CREATE12x12**

If the cell file newcell.cel does not already exist in the current directory when the command line above is executed, the startup command file new.cmd will be executed to initialize technology settings, then the commands in create12x12.cmd will be executed. If no errors are encountered, the editor will save the cell newcell in the current directory then close without the need for any user interaction.

See the AllCells utility description on page 378.

This option is particularly useful when you need to perform the same action in many cells. You can create a batch file with many ICED command lines each specifying a different cell name but using the same EXIT=*file_name* option. The editor will open and close once for each cell. The AllCells utility will automatically create such a batch file for all cells in a particular cell library.

The command file specified with the EXIT option can contain any valid commands. This includes the creation of new geometry, commands that require user interaction (such as the digitization of coordinates), export commands, and all of the extra command file commands described in the Command File Programmer's Reference Manual.

ICED=s_meg

Use special memory management for huge database.

See an overview on memory management on page 87.

This option is used to specify an alternative memory swapping method, similar to the ICED32 for DOS method of virtual memory management. This option is only useful for very large databases. Use this option only when 1.5 Gigabytes of storage is not large enough for your database, since this method is far less efficient (i.e. slower) than the default memory management method. Even very large chips rarely require this type of memory management unless the design has been flattened (i.e. the hierarchically nested cells have been replaced with their components.)

The maximum size of the database using this option is 60 Gigabytes. Specify the maximum database size in Megabytes. The value for *s_meg* (for **S**wap file **M**egabytes) must be an integer in the range 8:60000.

To improve the efficiency of this method, use the **RAM** command line option.

The memory management method specified with the ICED option is very much less efficient than using Windows virtual memory alone when Windows virtual memory and the special page swapping thrash. To reduce thrashing you can use the *RAM=p_meg* option. (The RAM keyword must be used **after** the ICED keyword.) Using *RAM=p_meg* tells ICED™ to behave as if it could use *p_meg* Megabytes of physical memory (i.e. the actual RAM chips in your computer).

Example:

ICED MYCELL ICED=10000 RAM=340

The command line above instructs ICED™ to create a swap file that can store up to 10000 Megabytes (approximately 10 Gigabytes) of layout data. The RAM option refines the swapping method to assume that 340 Megabytes of physical RAM is available. This is a good value to use initially if your system has around 512 Megabytes of physical memory. This preserves about 1/3 of storage for the system, but prevents ICED™ from thrashing between the Windows swap file and the special swap file set up for the layout editor.

The **TMP**

command line option controls where the swap file will be created.

Use of the ICED parameter requires the creation of **very** large swap files in the temporary directory. You must make sure that the temporary directory (Q:\ICWIN¹¹\TMP by default) is located on a drive with plenty of free space. If not, use the TMP command line parameter to create the temporary directory on a drive with more space.

Use the **WINDOWS** command line option (or no memory option) instead of this option when your database is not huge.

The WINDOWS command line option controls the maximum size of the database when you want to use the default memory management method. This default method of memory management is limited to about 1.8 Gigabytes of data by the Windows operating system. If neither the WINDOWS nor ICED options are used in the command line, the default is equivalent to WINDOWS=200 Megabytes.

The ICED and WINDOWS parameters are mutually exclusive. You should not specify more than one of these parameters.

¹¹ Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

LEAVE=*file_name****Execute command file and leave***

Use the EXIT command line option instead of the LEAVE option if you always want to save the cell file. Use the QUIT option instead to avoid overwriting the cell file. Use ALWAYS to leave the editor open after the command file is complete.

Example:

This option is very similar to the EXIT command line option. It executes a command file as soon as the layout editor opens. The layout editor will close automatically after the command file is finished. If the command file has no errors, the editor will close automatically with a LEAVE command. If the geometry in the cell has changed, it will be saved. However, if the command file resulted in no changes to the geometry of the cell, the cell file will not be overwritten.

If no extension is given in *file_name*, .CMD is assumed. If *file_name* is not stored in a directory on the command file search path, you must supply the directory path to the file in *file_name*. If you indicate a file that does not exist, ICED™ will warn you with an error message in the journal file and then close the editor without saving the cell.

ICED MYCELL LEAVE=SWAPIT

See page 48 for details on the command file search path.

This command line will open the cell MYCELL and execute the commands in the command file SWAPIT.CMD. Since no directory path is included with the command file name, SWAPIT.CMD must exist in the working directory or in one of the directories in the command file search path.

See page 99 for information on the journal file.

If SWAPIT.CMD contains no errors, the editor will close with a LEAVE command. If SWAPIT.CMD does contain an error, then the remaining commands in the file will not be executed and the editor will close with a JOURNAL command. The cell file will not be saved. Look for the error message at the end of the journal file and correct the command file. Delete or rename the journal file before re-executing the ICED command line to avoid the automatic recovery option.

See the AllCells utility to build a batch file that contains ICED command lines to execute a command file in all cells of a library.

LIMIT=ON

Force view-limit mode to reduce display complexity

The LIMIT option is a better method to speed the initial display than the obsolete DEPTH command line option.

When you edit a large dense cell, ICED™ may take a while to draw the initial display. When you use ICED™ to edit this type of cell, the time it takes to draw the screen can be greatly reduced by using the LIMIT=ON command line parameter. This forces the VIEW LIMIT mode to on before the initial display is drawn.

The VIEW LIMIT mode is fully described on page 363. Basically it reduces the complexity of the display when the design is zoomed out to a fine scale. Shapes drawn in nested cells and shapes smaller than a minimum number of pixels will not be displayed.

The default VIEW LIMIT mode is off for all new cells. This prevents confusion when new users import a cell with UnStream or UnCIF and open it for the first time. They expect to see all of the geometry.

However, once users are familiar with the VIEW LIMIT mode, they usually prefer to open large cells with this mode turned on. When the entire design is drawn, the complexity of the display is reduced. However, once they zoom in on a smaller area of the design, the design is automatically drawn in complete detail.

Example:

ICED STREAMCELL LIMIT=ON

Let us assume that STREAMCELL is a large cell just imported with the UnStream utility. The initial view window when the editor opens will include the entire design area. If this design was drawn in complete detail, the initial display may take a few moments and it will be so dense that it is difficult to see anything. The LIMIT=ON option will reduce the complexity of the initial display considerably making it easier to make sense of the data and drawing the screen much faster. When the view window is zoomed in on a smaller area, the view limit will become inactive and all detail will be displayed.

The VIEW LIMIT mode can be enabled in the startup command file, however using this option in addition to the setting in the startup command file is still a good idea. The initial display of a new cell is drawn before the startup command file is processed.

LOG=*cmd_count*

Control journaling

See page 99 for more information on the importance of the journal file.

This parameter sets the number of commands ICED™ will execute before logging them into the journal file. *cmd_count* must be an integer in the range 1:10. The default is 1.

By default, when ICED™ executes a command entered from the keyboard or a menu, it writes the command in a journal file as soon as it is completed. The file is opened and closed after each command. By using the LOG parameter, you instruct ICED™ to wait until *cmd_count* commands have been executed (or the command buffer is full) before writing to the journal.

Normally, using this parameter will result in only a slight improvement in performance. However, if the system crashes, up to *cmd_count* commands can be lost. Therefore, we recommend against using this parameter unless the journal file is being written on a removable drive or over a slow network.

The journal file is always written in the working directory, i.e. the same directory as the root cell.

MAXIMIZE=(YES | NO)

Control size of editor window.

This option allows you open the editor in maximized mode, i.e. the layout editor will fill your entire screen. The default is to open the editor in a smaller window that allows you to interact with other windows at the same time.

Example: **ICED MYCELL MAX=YES**

Change the size of an unmaximized window by dragging a corner with the mouse.

The command line above will open the editor window in maximized mode. You can switch to other windows with any of the standard Windows methods. The button next to the close button in the upper right corner of the window can be used to switch to normal (i.e. unmaximized) mode.

MENU=(*menu_name* | *)

Override menu

The **MkMenu** utility can be used to create customized menus.

This parameter is used to load an ICED™ menu. (The ICED™ MENU command can be used in the startup command file instead of using this parameter.)

When you exit a cell, ICED™ stores the name of the menu you were using in the cell file. ICED™ then tries to load this menu when you reopen the cell. Under certain circumstances this behavior can be inconvenient. The MENU parameter causes ICED™ to ignore any menu name stored in the cell file, and load menu *menu_name* instead.

For example, suppose a group of users all edit the same library of cells. One member of the group prefers to use a custom menu that is available only on his own computer. When he edits a cell, the name of the menu he is using is stored in the cell's environment database. If one of the other users edits the cell with no MENU parameter specified on the command line, ICED™ will not be able to find the menu file and the cell is loaded without a menu. However, if the other users use the MENU=M1 command line parameter, the menu file M1.MEN will be loaded instead of the menu file referred to in the existing cell file.

Learn about the auxiliary file search path on page 44.

Do not specify the menu file extension or path with *menu_name*. The file *menu_name*.MEN should already exist in an AUXIL subdirectory.

Example:

ICED MYCELL START=NEW.CMD MENU=M1

Using the MENU parameter above will cause ICED™ to ignore the menu reference stored in an existing cell file and load the menu file with the name M1.MEN. (This is the name of the menu file supplied with the ICED™ installation.) However, if you have launched ICED™ to create a new cell, the MENU parameter is processed first, loading M1.MEN. After the entire command line is processed, the NEW.CMD command file is processed. If the startup command file contains a MENU command, that command will replace the menu.

Example: **ICED MYCELL MENU=***

Using '*' for the menu name will load the default menu. As of this writing, the default menu is M1.MEN, but this may change with program updates. To always use the most current general menu available, use MENU=* on the command line in the batch file that launches ICED™.

NDIV=*ndiv_int*

Set divisions per user unit.

See the **QDIV** command line parameter if you must specify NDIV smaller than 100.

This parameter specifies the size of an ICED™ database unit. Most users should use the default value of 1000, which is virtually an IC industry standard. While ICED™ will accept any value of NDIV from 1 to 10000, values of less than 100 are not recommended. NDIV is an important parameter. Even if you intend to use the default value, we recommend reading this section carefully.

When you supply a coordinate in an ICED™ command, or when ICED™ displays a coordinate on the screen, the value is given in user units. You do not have to specify the size of the user unit until you are ready to make a STREAM or CIF output file. However, the vast majority of users think of user units as microns.

See the **ADD** command description for more details on how coordinates are digitized and stored.

ICED™ stores every coordinate as an integer number of database units. Therefore, the smallest distance ICED™ can resolve is 1 database unit. NDIV specifies the number of database units in 1 user unit.

$$\mathbf{1\ database\ unit = 1\ user\ unit / NDIV}$$

Coordinate values are stored as 32-bit integers. A 32-bit integer can represent numbers in the range from -2,147,483,647 to +2,147,483,647. However, ICED™ only allows coordinates in the range -250,000,000 to +250,000,000 database units. (This is a programming convenience that allows ICED™ to do computations on coordinates without checking for integer overflows.) The size of the database unit determines both the smallest feature ICED™ can represent and the largest area it can manage.

Example:

NDIV=1000

When this parameter (the default) is used, if 1 user unit = 1μ, then 1 database unit = 0.001μ. The smallest displacement ICED™ will be able to represent is 0.001μ. The largest chip it will handle is a square with sides no larger than:

$$0.001 \bullet 2 \bullet 250,000,000\mu = 50\ cm = 19.68\ inches.$$

PAUSE=*p_seconds* *Control how long editor settings are displayed*

The console messages are saved in the journal file.

When the ICED™ layout editor is launched, it first opens a console window to display messages about selected options and information on opened cells. After a pause to let you see the messages, the console window is replaced with the layout editor window. The duration of this pause is controlled by the PAUSE command line parameter.

When you terminate ICED™ and cell files will be saved, a similar pause will let you briefly see a report on the exact locations of the saved cell file(s). This pause is the same length as the pause at the beginning of the program.

The default pause is 3 seconds. Override the length of the pause by using the PAUSE keyword followed by the length of the pause expressed as an integer number of seconds in the range 0:60.

Three seconds is usually enough time to notice if something went wrong during initialization. To prevent the console window from being closed, you can press the <Esc> key to take as long a look at the initialization messages as required.

Example: **ICED MYCELL PAUSE=30**

Specifying a pause this long will give you plenty of time to look at the initialization messages. Press <Enter> or both mouse buttons to interrupt the pause and open the editor immediately.

Example: **ICED MYCELL PAUSE=0 QUIT=STREAM_EXPORT.CMD**

Avoiding the initial pause entirely with this command line can be useful if you do not find the messages of interest. It is also useful when opening the editor to perform a task with a command file and then quit without user interaction. The addition of PAUSE=0 to the command line will avoid an extra delay of up to six seconds.

QDIV=*ndiv_int*

Set divisions per user unit without warning

Use the **NDIV** command line parameter instead of this parameter unless you must specify NDIV smaller than 100.

This parameter is one of two ways to specify the NDIV parameter. Use the QDIV keyword instead of the NDIV keyword only when you need to specify NDIV<100 and you want to avoid a warning message.

Most users should use the default value of 1000 for NDIV, which is virtually an IC industry standard. While ICED™ will accept any value of NDIV from 1 to 10000, values of less than 100 are not recommended. This is due to the extreme granularity of coordinates that will result.

Read the NDIV parameter description on page 76 thoroughly before using the QDIV parameter.

QUIT=*file_name*

Execute command file and quit

Use the EXIT or LEAVE command line options instead this option to save the cell file.
Use ALWAYS instead to leave the editor open after the command file is complete.

This option is very similar to the EXIT command line option. It executes a command file as soon as the layout editor opens. With this option, the layout editor will close automatically with a QUIT command after the command file is successfully completed. The cell file will not be overwritten.

If no extension is given in *file_name*, .CMD is assumed. If *file_name* is not stored in a directory on the command file search path, you must supply the directory path to the file in *file_name*. If you indicate a file that does not exist, ICED™ will warn you with an error message in the journal file and then close the editor without saving the cell.

Example:

ICED MYCELL QUIT=DRCEXPORT

See page 48 for details on the command file search path.

This command line will open the cell MYCELL and execute the commands in the command file DRCEXPORT.CMD. Since no directory path is included with the command file name, DRCEXPORT.CMD must exist in the working directory or in one of the directories in the command file search path.

See page 99 for information on the journal file.

If DRCEXPORT.CMD contains no errors, the editor will close with a QUIT command. If DRCEXPORT.CMD does contain an error, then the remaining commands in the file will not be executed and the editor will close with a JOURNAL command. Look for the error message at the end of the journal file and correct the command file. Delete or rename the journal file before re-executing the ICED command line to avoid the automatic recovery option.

See the AllCells utility to build a batch file that contains ICED command lines to execute a command file in all cells of a library.

RAM=*p_meg*

Refine memory when ICED option is used

See an overview
on memory
management on
page 87.

Use this parameter only in combination with the ICED=*s_meg* parameter. The ICED option is used to force ICED™ to use a slower alternative method of virtual memory management for huge databases (>1.5 Gigabytes).

The memory management method specified with the ICED option is much less efficient than using Windows virtual memory. The editor will execute particularly slowly when Windows virtual memory and the swap file created by the ICED option thrash. To reduce thrashing you can use the RAM=*p_meg* option.

The RAM option will never improve the editor response time enough to compare to the response time of the editor using the default memory management. However, experimenting with this option may improve the response time slightly.

See the **ICED**
command line
option on page
67.

Specify *p_meg* as an integer number of Megabytes in the range 8:512. This is a limit on the amount of system memory ICED™ can use before using its special page swapping. RAM=*p_meg* limits ICED™ to *p_meg* Megabytes of physical memory (i.e. the number of **M**egabytes of actual RAM chips in your computer). The idea is to prevent ICED™ from requesting so much memory from the system than the system resorts to the system swap file for virtual memory. However you want to use as large a number as possible to utilize all physical memory and speed processing.

A good initial value for *p_meg* is 2/3 of the amount of physical memory on your computer. (This allows the system and other resident processes some processing space without resorting to the system swap file.) However, some experimentation will be required to find the optimum number given your needs, physical resources, and system demands.

Example:

ICED MYCELL ICED=20000 RAM=512

The command line above instructs ICED™ to create a swap file that can store up to 20000 Megabytes (approximately 20 Gigabytes) of layout data. The RAM option refines the swapping method to assume that 512 Megabytes of physical

RAM is available. This is a good value to use initially if your system's physical memory is $\frac{3}{4}$ Gigabyte or more. This preserves about $\frac{1}{3}$ of RAM for the system, but prevents ICED™ from thrashing between the Windows swap file and the special swap file set up for the layout editor.

Set RAM to a smaller number the next time the editor is opened if you suspect thrashing between the two types of virtual memory with your current values.

STARTUP=*file_name* *Execute startup command file on new cells*

See an overview on startup command files and an example on page 25.

Whenever ICED™ creates a **new** cell, it begins by executing the ICED™ commands in the file indicated by this parameter. The commands in this file typically initialize layer names and colors, grids, snap values, and other cell environment parameters.

Read more about the paths searched for command files on page 48.

You can specify the drive letter and path as part of the startup file name. If you omit the drive letter and path of the file, ICED™ will search all directories on the command file search path for the file. If you omit the file extension, ICED™ will search for *file_name*.CMD. If the file name contains blanks, you must surround it with quotes.

Example:

ICED MYCELL START=Q:\ICWIN\TECH\SAMPLES\NEW

The current startup command file can be executed at any time with the @* command in the editor.

This sample command line option will result in the commands stored in the file NEW.CMD (supplied with the ICED™ installation) being executed for every new cell.

You should take the time to carefully create the startup command file for your technology before creating any cells. **If you make changes to your startup command file, only new cells will use the new settings.** All existing cells will still use the obsolete settings from the previous version of your startup command file.

If you want to execute a command file every time the editor is opened, use the **ALWAYS** command line parameter instead of, or in addition to, STARTUP. (See an example of combining the STARTUP and ALWAYS options on page 116.) If you want to have ICED execute a command file and then terminate without any user interaction, use a **QUIT**, **EXIT**, or **LEAVE** command line option. (See an example of combining the STARTUP and EXIT options on page 66.)

TIME_OUT=*t_seconds*

Allow more time for security check

This option is intended only for the infrequent user who has trouble with the security check timing out before the system can respond. The TIME_OUT option was needed primarily with the old system of security keys, and is usually not required.

The program that verifies that the security key is present on your computer will time out by default after several seconds. The system must respond to a request for information within this time for the security check to pass and allow ICED™ to open.

On some systems, the default 10 second wait (20 seconds on NT) is not long enough for the security program (ICEDnAUX.EXE) to pass. In this case, use the TIME_OUT keyword and specify *t_seconds* as an integer number of seconds in the range 10:60.

Example: **ICED MYCELL TIME=30**

Note that the TIME_OUT keyword can be shortened to any unambiguous abbreviation. This is true of all ICED™ keywords.

TMP=*path_name*

Specify temporary directory

ICED™ can create several temporary files that are deleted automatically when you close ICED™ normally. By default, ICED™ puts these files in the directory Q:\ICWIN¹²\TMP. This directory will be created if it does not exist. The TMP parameter allows you to tell ICED™ to store temporary files in a different directory.

There are two important cases where you may want to use the TMP parameter.

- If you use the ICED command line parameter to have ICED™ manage its own virtual memory, the size of the files created in Q:\ICWIN\TMP can get very large. If disk partition Q: does not have sufficient space you may want to tell ICED™ to store its temporary files on another drive.
- If you are running ICED™ on a network, several users may be using the program at the same time. If you have set up a method for sharing the ICED™ distribution files, you should use the TMP parameter to make sure that each user has his own TMP directory. Otherwise, two copies of ICED™ may try to store data in the same temporary file. This will result in the data being corrupted.

Example: **ICED MYCELL TMP=D:\TEMP**

Using this parameter on the ICED™ command line will force ICED™ to use the TEMP directory on the D: drive to store all temporary files.

¹² Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

VIEW_ONLY=(ON | OFF)

Set view-only mode

This parameter can be used to allow you to view cells with ICED™ instead of editing them. You cannot save modified cells if the view-only mode is being used. When the default of VIEW_ONLY=OFF is used, ICED™ behaves normally, and you will be able to save changes.

Example: **VIEW_ONLY=ON**

When you use this command line parameter, ICED™ will refuse to process an EXIT command. No cell files can be modified, but other commands that create export files (e.g. DRC, STREAM and PLOT) are still enabled. You would usually use the QUIT command to close the editor in this mode.

If you edit a cell in this mode inadvertently, and you need to save changes, you can recover work done during the session by using the JOURNAL command to close the layout editor. You can then recover the work done with the automatic recovery feature after you have removed the VIEW_ONLY=ON option from the command line. Refer to the example on page 211 and the information in "Journaling and Data Recovery" in Appendix B of this manual.

WINDOWS=w_meg

Limit memory

See an overview
on memory
management on
page 87.

By default ICED™ makes use of Windows virtual memory. Unless the ICED or WINDOWS parameters are defined on the command line, ICED™ will allocate a 200 Megabyte block of virtual memory for the database. (This is enough memory for almost all hierarchical designs, including large chips.) By pre-allocating the memory, the data can be stored in a single large block of memory. There is almost no cost to this allocation except that some operating systems (e.g. NT) make sure that there is enough disk space for the swap file to grow to accommodate the request.

If you have limited disk space, you may see ICED™ fail with a message similar to the following:

"Error reserving virtual memory for data pages. System reports

In this case, use the WINDOWS keyword with a value for *w_meg* smaller than 200. 100 Megabytes is usually enough for average databases. Add the following string to the ICED.EXE command line in your batch file:

Example:

WINDOWS=100

If you have a large flat database, you can specify *w_meg* as a number larger than 200. Specify *w_meg* as an integer number of Megabytes in the range 8:2000.

If 2000 Megabytes (2 Gigabytes) is not enough space for your database, use the ICED parameter instead to force ICED™ to use an alternative system for virtual memory.

The ICED and WINDOWS parameters are mutually exclusive. You should not specify more than one of these parameters.

Memory Management

This is an expanded description of the WINDOWS, ICED, and RAM command line options.

ICED™ offers two methods of controlling memory available to the layout editor. The default method is to use the virtual memory management available through the Windows operating system. This is the most efficient method. However, this method is limited to layout databases of approximately 1.8 Gigabytes. The other method is to have ICED™ create its own swap file

When you do not override the default memory settings sent to the layout editor, Windows virtual memory will be used. Since ICED™ is more efficient when it allocates virtual memory as a large block, it allocates memory at the start of the program. It will allocate 200 Megabytes by default. Even very large chips rarely require any additional memory unless the design has been flattened (i.e. the hierarchically nested cells have been replaced with their components.)

For the majority of users, this is a large enough number to supply plenty of memory for typical layout databases, and a small enough number that typical computer systems can provide the required resources. However, if your computer cannot supply 200 Megabytes of virtual memory, or if your design requires more than this amount, you can override the defaults with command line options.

Refining Memory for Computers with Limited Resources

If you have a limited amount of physical memory or disk space on the drive that contains your computer's swap file, ICED™ may not be able to allocate the 200 Megabytes required by the layout editor's defaults. When this is the case, you may see ICED™ fail with a message similar to the following:

"Error reserving virtual memory for data pages. System reports

You can override the layout editor's initial memory allocation with the WINDOWS command line option. The amount of memory is supplied in

Megabytes. When computer resources are limited, supply a value smaller than 200. 100 Megabytes is usually enough for average databases. While values as small as 8 Megabytes are permitted, we recommend that you do not limit ICED™ to that small an amount of memory unless absolutely necessary.

Example: **WINDOWS=100**

See page 86 for a complete syntax description of this command line option.

Adding this string to the ICED.EXE command line in your project batch file will limit the editor to around 100 Megabytes of memory.

Memory Management for Large Databases

It is unlikely that you will need more than the default amount of virtual memory if your design is hierarchically nested. However, if you need to access a large flattened design (i.e. all shapes are in the main cell), 200 Megabytes of virtual memory may not enough to load your design. If you think 1.5 Gigabytes will be enough, you can use the WINDOWS command line option to increase the initial memory request.

Example: **WINDOWS=1000**

You can add this string to the ICED.EXE command line to increase the initial memory request to approximately 1 Gigabyte.

The WINDOWS option allows you to supply a value up to 2000 Megabytes, roughly equivalent to 2 Gigabytes. However, values this high may place stress on your operating system especially your system swap file. If you need this much memory, we recommend using the alternative virtual memory management options described below instead of that specified by the WINDOWS option.

Memory Management for Huge Databases

The ICED and RAM options control a swapping method similar to the ICED32 for DOS method of virtual memory management. Use these options only when

1.5 Gigabytes of storage is not large enough for your database, since this method is less efficient (i.e. slower) than the default memory management method.

These options are specified on the ICED.EXE command line in the batch file to launch the editor. Do not use the WINDOWS option if you use the ICED option.

See page 67 for a complete syntax description of the **ICED** option and page 80 for the **RAM** option.

The total size of virtual memory is specified with the ICED option. The maximum size of the database using the ICED option is approximately 60 Gigabytes.

The memory management method specified with the ICED option is very much less efficient than using Windows virtual memory alone when Windows virtual memory and the special page swapping thrash. To reduce thrashing you can use the RAM option to indicate to ICED™ how much physical memory is on your computer. The idea is to prevent ICED™ from requesting so much memory from the operating system than the system resorts to the system swap file for virtual memory. However you want to use as large a number as possible to utilize all physical memory and speed processing.

Example: **ICED=60000 RAM=512**

This pair of command line options provide the largest amount of memory available in ICED™. This allocates approximately 60 Gigabytes of virtual memory and 512 Megabytes of physical memory. To use options this large, be sure that you have at least 700 Megabytes of physical memory available on your system, and plenty of free disk space on the drive where the temporary swap file(s) will be stored.

The **TMP** command line option controls where the swap file will be created. See page 84.

Use of the ICED parameter requires the creation of **very** large swap files in the temporary directory. You must make sure that the temporary directory (Q:\ICWIN¹³\TMP by default) is located on a drive with plenty of free space whenever you have a very large database. If not, use the TMP command line parameter to create the temporary directory on a drive with more space.

¹³ Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

Executing Batch Files on a Windows Platform

You can use the CD command in the project batch file to make the working directory the current directory before the ICED.EXE command line.

You can execute a batch file on a Windows operating system with several methods.

- Open a console window with the ICED icon created on your desktop by the installation and then simply type the name of the batch file at the console prompt. (You may want to make the working directory the current directory first by typing a CD command in the console window.) You can pass arguments on the batch file command line with this method.
- You can type the batch file command line in the window opened by the task bar option Start->Run. (In this case the CD command to make the working directory the current directory should be located in the project batch file.) You also can pass arguments on the command line with this method.
- If you routinely open a file manager (like Windows File Explorer), you can simply double click your mouse on the name of the batch file. On some operating systems you can pass arguments with this method. These operating systems allow you to create a .PIF file to modify the properties of the command line and pass arguments.
- You can drag the batch file icon from Windows File Explorer to your desktop with the right mouse button. You can type arguments on the batch file command line and specify the working directory by modifying the shortcut properties.
- Some versions of the Windows operating system (e.g. Windows 95, 98, and XP) allow you to assign a program to a file extension. When you select a file by double clicking it, the program is executed using the file name as an argument.

Therefore, you can assign your project batch file as the program for files with a .CEL extension. In this case, when you double click on a cell file with Windows File Explorer, the project batch file will execute with the name of the cell as an argument. With this method you do not need to open a console window explicitly to run ICED™.

If you want to pass ICED™ command line parameters (other than the cell name) into the batch file as arguments, you may need to add the BATCH keyword to the command line. See page 60.

Using ICED™ to Perform Tasks with no User Interaction

A batch file can be written to open the layout editor, execute some specific commands, and then close automatically. This is useful for many routine tasks such as the export of design data. This method allows you to repeat the exact same procedure each time without the need for user interaction.

There are five different ICED.EXE command line options that will automatically execute a command file after it opens. The first three in the list below close the editor automatically after the command file is complete.

	ICED.EXE Command line option	Pg.	Use
Use only one of these	EXIT	65	Execute command file then close editor with an EXIT command. This will overwrite the cell file.
	LEAVE	69	Execute command file then close with a LEAVE command. This will overwrite the cell file only if changes have been made to the geometry of the cell.
	QUIT	79	Execute command file then close with a QUIT command. This will not overwrite the cell file.
	ALWAYS	58	Execute command file and leave editor open.
	STARTUP	82	Execute command file only for a new cell and leave editor open.

Figure 10: ICED.EXE command line options that automatically execute a command file

Even when you use one of the last two methods, if the command file contains a termination command (i.e. EXIT, LEAVE, or QUIT) then the editor will close automatically. (Except in the case where the command file uses the termination command only to return from editing a child cell.)

When an error is encountered in the command file, the rest of the command file is not processed. Only errors encountered in the command files using the ALWAYS or STARTUP options will be reported on the screen. The error messages are always recorded in the journal file.

When an error is encountered in the command file using the EXIT, LEAVE, or QUIT options, the editor automatically closes with a JOURNAL command and the cell file is not saved.

The command file can contain an export command like DRC (for export of data to the DRC Design Rules Checker or to the LVS Layout Vs. Schematic checker), CIF or STREAM (to create layout files for other processing), or PLOT (to create a printout of your layout data.) Alternatively, you can use commands that modify the cell geometry such as the SWAP command to change subcell references or layers of certain components.

New with version 4.82 is the ability to combine the STARTUP command line option with one of the other options in Figure 10.

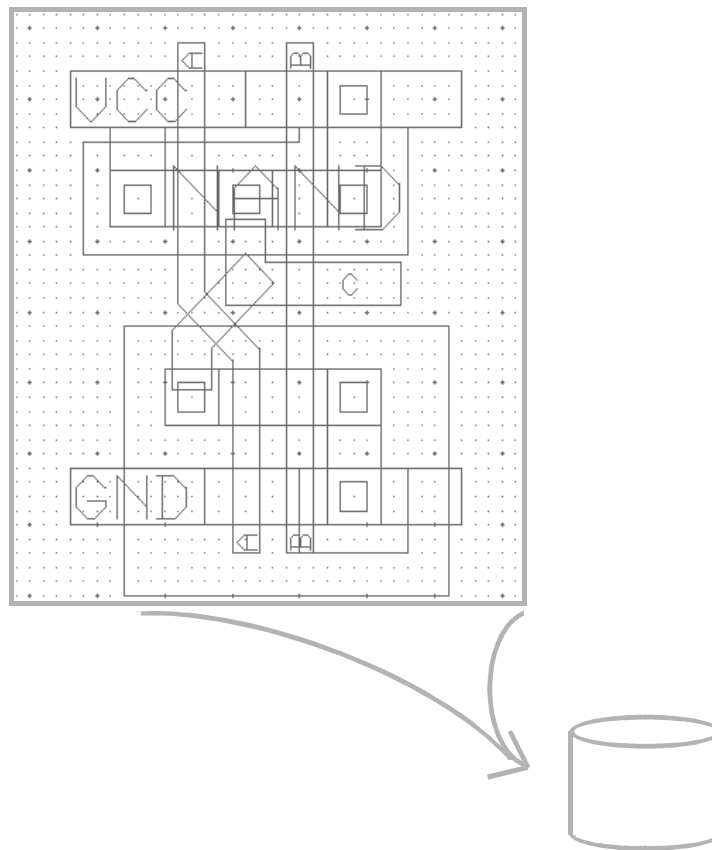
You can create cells without user interaction with these options. Combining the STARTUP and EXIT commands allows you to execute two separate command files. The startup command file can be used to initialize technology and project settings, then the command file specified with the EXIT option can contain commands that create components. The editor will close automatically after saving the cell file since the EXIT option is used. (See an example on page 66.)

You are not prevented from using commands in these files that require user interaction. Any editor command can be used, including those that require that the user to digitize coordinates. All of the extra commands documented in the Command File Programmer's Reference Manual are also supported.

The AllCells utility can be used to create a batch file that will execute a command file in each of the cells of a cell library.

See two complete examples of batch files without user interaction in the appendix beginning on page 449.

Overview of the ICED™ Layout Editor



The ICED™ Layout Editor Window

Do not change the resolution of your monitor or the video mode while ICED.EXE is running.

Change the size of an unmaximized window by dragging a corner with the mouse.

When the ICED™ layout editor is launched, it first opens a console window to display messages about selected options and information on opened cells. After a pause to let you see the messages, the console window is replaced with the layout editor window. This window will display the current view of the selected cell. You modify the cell or the view with commands typed on the prompt line near the bottom of the screen. You can also execute commands by clicking menu choices.

These commands affect only the current cell. You can open another cell during the session with a subcell edit command, but only one cell is open for modification at a time. There is an option to leave parent cells displayed while you edit a nested cell, but there is only a single view window and a single cell that can be modified at any given time.

Components and Cells

The basic building blocks used by the ICED™ editor are components. ICED™ components include **boxes, polygons, wires, lines, text, cells, and arrays**. More complex entities like circles, arcs, and rings are based on these basic components. These circular entities are stored as polygon or wire components but are generated with commands that create components with precise geometric properties.

Cells can also be created with the **GROUP** or **EDIT** commands.

Cell and array components are collections of other components. A cell is created when you place components in your drawing and then save it. Once you have created a cell, you can add that cell to another cell as a component. You can also add arrays of cells. A cell or array component can be manipulated just like any other component.

A cell or array, added to another cell as a component, is said to be **nested** inside the other cell. A nested cell is often referred to as a **subcell**. The cell you launch ICED™ to edit, is referred to as the **root cell**. You can edit other cells during an ICED™ session using the EDIT, P_EDIT, or T_EDIT commands. A cell edited with one of these subcell edit commands is referred to as a **child cell**. The cell you were editing when you issued the edit command is referred to as the **parent cell**.

Geometric Database vs. Environment Database

Each ICED™ cell file contains two databases. The first is the **geometric database** that stores all data specific to components. Component types, layer numbers, and coordinates are stored here. The selection status of components is also saved here, including flags which indicate that components are blanked (made invisible) or protected from changes.

The environment database for a new cell is defined with the startup command file. See page 25

The **environment database** contains data related to the cell as a whole. Most technology related settings are stored here (e.g. layer definitions and the resolution grid). Display parameters, including color definitions, view settings, and menu file references, are also stored in the environment database.

When you launch ICED™ to open an existing cell as a root cell, both databases for that cell are loaded. However, for any child cells, only the geometry database is loaded. For instance, if you have assigned the color red to layer 1 in the root cell, and assigned the color blue to layer 1 in a nested cell, components on layer 1 in the nested cell will appear red.

The environment database of the root cell is the only environment database loaded during an ICED™ session. If another cell is modified and saved during the session, the environment database of that cell is replaced with the environment database of the root cell. In the example above, if the nested cell that had color blue assigned to layer 1 is saved, that information is lost and layer 1 will be drawn in red even if that cell is loaded as a root cell in another session.

For this reason, it is best to keep the environment database of all cells in a project identical. This is accomplished with a startup command file, a subject covered in detail beginning on page 25.

Remember that if you change layer settings, colors, display grids, etc. in an edit session, those changes are stored in all cells tagged for saving during the session.

Creating New Cells

When you specify a cell name on the ICED.EXE command line and that cell file does not exist in the current working directory, then ICED™ will create a new empty cell. Restrictions on the names of new cells are indicated below. The startup command file will be executed to create the environment of the new cell.

If you try to create a new cell with the same name as one already in a cell library, you will be warned. This is to prevent you from accidentally replacing a cell you are already using with a new cell in a different library.

Cell Name Restrictions

The **Del2Cel** utility converts old cell files named with the *@ustring.DEL* format to files that use the long cell name as the name of the cell file.

Cell names may be from 1 to 32 characters long.

The following special characters **are** allowed in cell names:

#

\$

_ (underscore NOT hyphen)

Other non-alphanumeric characters are NOT allowed.

Blank spaces are NOT valid in cell names.
--

Opening Existing Cells

See page 20 for a method of opening a cell when you cannot use its directory as the working directory.

When the cell you specify on the ICED.EXE command line already exists in the working directory, its cell file is read into the layout editor. The cell environment database stored in this root cell file is used to create the layout editor environment. The startup command file defined with the STARTUP command line option is not executed. If a command file is defined with an ALWAYS option, it will be executed.

As the cell file is read, the editor must search for cell files for subcells referred to in the root cell file. It will search for cell files in each of the cell libraries defined on the cell library search path described on page 45. If it cannot find a cell file for a subcell, you will receive an error message as described below and the editor will not open.

Missing Cell Error Messages

If the layout editor reports an error message similar to the following instead of opening your cell, this means that cell files for subcells of the root cell cannot be found.

******* There are 2 missing cells *******

These messages must be resolved before you can edit or view the cell. If you get an error message similar to the one above, it is probably due to one of three reasons:

- Some cell libraries are omitted from the list in the ICED_PATH environment variable. See page 45.
- Some cell files are still in the *@string.DEL* format used by older versions of ICED™ for cells with long names. See the Del2Cel utility on page 378.
- Cell files are actually missing. This is not uncommon when opening cells imported from Stream or CIF files. Contact the person who gave you the import file to resolve these types of problems.

When Cell Data Is Saved in Cell Files

When you terminate the editor, the layout of all cells flagged for saving during the session is saved in cell files. (We will cover how cells are flagged for saving on page 100.) The current environment (e.g. layer names, colors, grid and window parameters) is stored in each of the cell files.

See page 45 to learn about when cell files are saved in other cell libraries.

No cell data is stored on disk until you exit the layout editor.

Cell files are usually stored in the working directory. The name of the cell file will be *cell_name.CEL*, where *cell_name* is the name of the cell.

See **Journaling and Data Recovery** in **Appendix B** to see a complete description of recovering cell backups.

When you save an existing cell, ICED™ will make a backup of the cell file before it is overwritten. This backup cell file can be used to recover from errors. If the cell file is named *cell_name.CEL*, the backup file name will be *cell_name.CL1*.

The complete series of steps for saving each cell file is listed in Appendix B. If one of these steps fails (e.g. you run out of disk space), the layout editor issues a "SAVE ERROR" message, skips the remaining steps for that cell, and goes on to the next cell. (The error messages are displayed on the screen and recorded in the journal file. This file will be described on page 99.) It is rare for problems to occur during these steps, but if you get a warning message at this stage see the information on page 440 in Appendix B.

Commands and Command Files

Components are created and manipulated with commands. There are also commands to manipulate the environment database. The ICED™ Command Reference chapter beginning on page 109 lists each command. The commands can be typed at the keyboard or selected from a menu.

An introduction to command files is found on page 14.

Commands can also be typed in a file called a command file. The startup command file is an example of a command file. When a command file is executed, the commands in it are executed as though they were typed at the keyboard.

An entire tutorial is devoted to command files in the Classroom Tutorials Manual.

You can execute a command file automatically when the editor opens with any of the command line options described on page 91. When the editor is already open, you use the `@file_name` command to execute a command file. (Look at that command description on page 115 for more details on executing command files.) You can also execute a command file by selecting it from the lists offered by the menu option 3:@%.cmd¹⁴. (In this case the command file must use a .CMD extension and be stored in a directory on the command file search path.)

To learn about command files in depth, read the Command File Programmers manual.

(If you name the a command file with a "_" prefix, the file will not appear on the menu lists, but it can still be found by the other methods. This prevents command files intended only as "helper" command files from cluttering the menu lists. "Helper" command files are called by other command files to execute specific tasks like value validation.)

Journaling

ICED™ records every command executed during an edit session in a journal file. The name of the journal file will be *cell_name*.JOU, where *cell_name* is the name of the root cell. The journal file is stored in the working directory. This file is written to disk as the commands are executed. When you terminate ICED™ normally with the EXIT, QUIT, or LEAVE commands, the journal file is renamed to *cell_name*.LOG.

See Journaling and Data Recovery in Appendix B.

If your system crashes, or you use the JOURNAL command to exit ICED™, the journal file will not be renamed. The next time you edit the cell, ICED™ will notice that *cell_name*.JOU still exists, and the program will offer to perform an automatic recovery of all changes made before the crash.

¹⁴ When we discuss menu options, the number before the colon refers to the top-level menu number. (You switch between top-level menus with the right mouse button.)

The journal files are written in ASCII in ICED™ command syntax. They can be used to recover work in the event of a system crash or editing mistake. Journal files are also useful to determine the exact syntax of commands generated when you use the menus to execute a command. You can edit journal files to create command files for execution in future ICED™ sessions.

Terminating ICED™

See the **EXIT** command description for a more complete comparison of termination commands.

As a last resort, you can close the editor with the 'X' button at the top right corner of the window. This executes the Journal command.

You can finish editing a cell with one of four different commands: EXIT, LEAVE, QUIT, or JOURNAL. When you are editing the root cell, these commands will terminate ICED™ completely. If you have used the EDIT, P_EDIT, or T_EDIT command to edit a child cell, you will be returned to the parent cell when you use the EXIT, LEAVE, or QUIT command. The JOURNAL command always terminates the edit session immediately.

Cell files are not saved to disk until you exit the layout editor. When you exit from editing a nested cell, the cell is only flagged in memory. The cell file is not saved until the layout editor terminates completely. The EXIT command always flags the current cell for saving. If you use the QUIT command to terminate editing a cell, that cell will not be flagged for saving. In this case, the cell file and cell backup file will not be overwritten.

The LEAVE command flags a cell for saving only if its geometry has been modified. If no changes were made to the geometry of the cell, the cell file will not be flagged for saving. This is the best command to use routinely since cell backups will not be lost when you have made no changes to a cell.

When the EXIT, QUIT, or LEAVE commands are used to finish editing the root cell, ICED™ will terminate. The flagged cell files will be saved at this time. (See Appendix B to learn the specific steps followed when saving cell files.) The journal file will be renamed to *cell_name*.LOG at this time.

If your system crashes, or if you use the JOURNAL command to terminate ICED™, no cell files will be saved and the journal file will not be renamed. This will automatically result in ICED™ offering you the option of recovering your lost work the next time you edit the same root cell.

Syntax Conventions

[]

KEYWORD

()

< >

Manual Notation

The command and utility references use the following conventions when discussing the syntax of ICED™ commands and utilities:

KEYWORD Bold type in the syntax section will be used to indicate the command name keyword. Most keywords require only the first several letters of the keyword to be typed, as long as the letters result in a combination unique to ICED™. For example, **F** may be typed instead of **FILL** as there are no other commands which begin with the letter F.

(**KEYWORD A** | **KEYWORD B**) Parentheses will be used to indicate that a choice of parameters is **required**. The options will be delimited with the | character. Exactly one of the options must be used. Do not type the parentheses in the command.

Example:

MERGE (WIRES | LINES | POLYGONS)

This example shows the syntax for the MERGE command. Exactly one of the three keywords: WIRES, LINES, or POLYGONS, must be used. The following example demonstrates how you could type the command:

Example:

MERGE POLYGONS

Like all ICED™ keywords, these keywords can be abbreviated by using the first several letters of the keyword as long as the combination is unambiguous.

Example:

MER POLY

This example would execute the same command as the previous example.

[KEYWORD A | KEYWORD B] Square brackets indicate that the keyword or parameter is optional. If a choice of keywords is valid, they will be delimited with the | character. Either none or exactly one of the options is valid. When one of the options is a default, it will be underlined. Do not type the brackets in the command.

Example:

STREAM (MAP_LAYERS | ARCHIVE) ...
... [FULL | ROOT | NOLIBS] ...

This is a portion of the syntax for the STREAM command. One of the keywords, MAP_LAYERS or ARCHIVE, must be used. One of the keywords, FULL, ROOT or NOLIBS, may be used. If you type the command this way:

Example:

STREAM MAP_LAYERS

The FULL keyword is used by default.

In the syntax section, parameters which are not enclosed in square brackets are **required** by the command.

parameter value

Lower case italic type will be used to indicate where a value should be entered in a command. A value could be a number or a string. The valid values for the parameter will be indicated in the description.

Example:

```
COLOR color_id [ NAME=color_name | NO_NAME ]...  
... [ PALETTE=red_value, green_value, blue_value ] ...
```

This portion of the COLOR command syntax demonstrates several notation concepts. The *color_id* parameter is required since it is not enclosed in brackets. It represents either a color name or number, as indicated in the command description. The NAME or NO_NAME choice of parameters is optional, if you use neither, the name value of the color will not be changed. If you use the NAME parameter, use a string for *color_name*. The PALETTE parameter is also optional. When you do use it in the command, you must supply three integers which will be used as the color definition.

Example:

```
COLOR 8 NAME="DARK_BLUE" PALETTE=10,10,30
```

In this example, the number 8 is used for *color_id*. The string DARK_BLUE is assigned to this color as its name. The three integers in the PALETTE parameter define how the color is represented. The commas between the numbers are optional.

Commas are treated as white space characters. White space characters are characters that separate fields but have no other meaning. All white space characters are translated to blanks as the command is read. They are used to make commands more readable. Parentheses, tabs, and equal signs (=) are also treated as white space characters. Quotes are usually optional around strings. The above command could also have been typed in the following way:

Example:

```
COL 8 NAME=DARK_BLUE PAL (10 10 30)
```

The underscore (_) used in many keywords is also optional. It is provided to make keywords and names more readable. The underscore is stripped from keyword or name before the command is executed. The following two commands are equivalent:

Example:

COLOR 8 NAME=DARK_BLUE
COLOR 8 NAME=DARKBLUE

Layer and color names can also be abbreviated by using the first several letters of the name as long as the combination is unambiguous.

Example:

COLOR DARKB PAL (12 12 30)

If one of the commands above was used to assign the name DARK_BLUE to color 8, the command above would modify that color with the indicated palette even though the entire name was not typed. However, if another color begins with the same letters (e.g. DARK_BROWN) then the command would fail with the message "Ambiguous color name".

- a:z Ranges of possible values will be indicated by a colon between the lowest valid value and the highest valid value.

- ... Three dots at the end of a line of sample code, or in the syntax section, indicate that the command is continued on the next line. Three dots will also precede the continuation on the next line. The dots should not be included when the command is typed.

- <key> The less-than < and greater-than > symbols indicate that one specific key from the keyboard should be typed. For example, <F1> is used for the F1 key on the keyboard, **not** the letter F followed by the number 1.

- Q:\ICWIN When this manual refers to the directory path Q:\ICWIN you should replace it with the drive letter and directory path you specified when you installed ICED™. For example, if you installed ICED™ on drive C: in the directory IC, you should use C:\IC to replace Q:\ICWIN in the examples.

Layer Lists

See the **BLANK** command for examples using layer lists.

There are several ICED™ commands where a list of layers can be used as a parameter. Any combination of the following rules can be used when creating a layer list.

Layer Numbers

A single layer number is always valid as a layer list.

Layer Names

A single layer name is always valid as a layer list, as long as the name has previously been assigned to a layer number with a LAYER command. The list of layers which have names assigned to them can be shown with the TEMPLATE command. When using a layer name, you can assume that the name will be replaced with the layer number as the command is processed.

You can also abbreviate a layer name as long as the abbreviation does not match more than one layer name.

The Range Operator :

A colon (:) can be used to define a contiguous range of layers.

Example: **1:10**

This example represents layers 1,2,3,4,5,6,7,8,9, and 10.

When a colon is used in a layer list, an asterisk (*) can be used to indicate that the lowest or highest valid layer number is meant. The lowest valid layer number is 0 and the highest is 255. When an * is used to the left of the colon, ICED™ replaces it with the layer number 0. When the * is to the right of the colon, it will be replaced with 255.

Example: *:*
 0:255

These two layer lists are identical. Both represent the list of all layers.

The Addition (+) and Subtraction (-) Operators

Layers, or ranges of layers, can be added to the layer list with a plus symbol (+) or subtracted from the list with a minus symbol (-). This allows you to create non-contiguous lists of layers. The list is processed from left to right. Any layer that is not already on a list which is being built, but is subtracted with the -, is ignored. (Note that one exception to this rule is covered below.)

Example: 1:10 + 15:17 -5 -50 -6:8 +50

This example will result in the list of layer numbers 1,2,3,4,9,10,15,16,17, and 50. Since layer 50 is subtracted before it is in the list, the -50 is ignored. Then when +50 is processed, layer 50 is added to the list.

Example: *:* -M1
 -M1

These two layer list are identical. Both result in the list of all layers except for the layer named M1. When a '-' is the first character in a layer list, ICED™ assumes that it is preceded with the list of all layers.

ICED™ Command Reference

ADD
SELECT
BLANK
ROTATE
CUT
GROUP

Overview

The ICED™ editor provides a rich set of commands to create and manipulate integrated circuit design data. There are also many commands that control the appearance and complexity of the display.

Several other commands used exclusively in command files are documented in the ICED™ Command File Programmer's Reference Manual.

The **ADD** command is used to add a component to a drawing. Since this command uses many parameters to define each component, the **USE** and **LAYER** commands are used to define defaults for these parameters. The **TEXT** command defines additional defaults for the ADD TEXT command. The **SPACER** command controls the appearance of the spacing guide available while adding wires, boxes, or polygons.

One of the most important parameters for the ADD command is the layer on which a component is added. The layer can correspond to the physical layer on the actual circuit. You can define names for layers such as M1 and POLY. You can also define layers for any special purposes you may have. The **LAYER** command sets the default layer for the ADD command. It is also used to define the appearance (color, fill pattern, etc.) and name of each layer. In addition, the LAYER command defines how each layer is treated on export to CIF (Caltech Intermediate Form) and Stream (Calma-GDSII) files as well how the layer appears on plots.

Other layer related commands are the **INITIALIZE** command that performs initialization of layer parameters, the **PATTERN** command that loads fill patterns from a file, and the **SWAP** command that can be used to change which layer components are on.

The UNSELECT command is described in the **SELECT** command description.

Many commands require that you select the components to be acted upon. The **SELECT** command is used for component selection. If you select components prior to executing a command, any number of SELECT and UNSELECT commands can be used to build a set of selected components. After the command is executed, the components can be unselected with an UNSELECT command.

If no components are selected when you execute a command that requires a selected component, ICED™ will automatically allow you to select a component. This is referred to as an embedded SELECT command. When an embedded SELECT is performed, the component will be unselected automatically after the command is completed.

You can hold the <Shift> key down during embedded SELECT commands to select more than one component.

The embedded SELECT process is very efficient for most commands. It allows you to perform edit commands with the minimum number of keystrokes or mouse clicks. However, when you need to perform operations on a set of components, the more flexible SELECT - EDIT - UNSELECT process can be very effective.

One form of the SELECT command will select components within a "near box" centered at the cursor. The **NEAR** command is used to set the size of this near box.

The **SHOW** command will report or export information about selected components.

There are two ways to prevent a component from being selected. The first is the **PROTECT** command which marks components as being unselectable. The second method is to use the **BLANK** command, which not only prevents components from being selected, but also makes them invisible.

Once components are selected, a variety of editing commands can modify them. A selection of components can be grouped together and transformed into a separate cell with the **GROUP** command. You can use the **COPY** command to copy components to a new location with options to mirror or rotate them. You can use the **MOVE**, **MIRROR**, or **ROTATE** commands to move components. (The MOVE command can also stretch sides or move individual vertices of a component.) The **DELETE** command removes selected components. The **SWAP** command can replace one set of cells with another or swap the layers of selected components. The **MERGE** command combines two components into a single component. The **CUT** command divides components along a cut line.

When components are created, edited, or moved, their coordinates can be restricted to lie on a grid. This grid is created with the **RESOLUTION**

command. A temporary, more restrictive grid can be created with the **SNAP** command. Neither of these grids is displayed on the screen. However, you can define a grid for display on the screen which emulates these grids, or any other grid, with the **GRID** command.

There are a number of other commands which control how information is displayed on the screen. The **COLOR** command will allow you to redefine screen colors as well as assigning names to colors for ease of use. The **BLINK** command will cause a color to blink on the screen. The **PATTERN** command can load fill patterns which are assigned to layers using the **LAYER** command. The **FILL** command then determines whether ICED™ will draw the fill patterns or only the outlines of components.

The **BLANK** command can be used to render selected components, cells, or entire layers invisible. The **OUTLINE** command controls how cell outlines are drawn on the screen. The **DISPLAY** command has a number of parameters that can simplify the display and speed screen refresh. When displaying large arrays, the **ARRAY** command can be used to control how many elements are displayed.

VIEW LIMIT can limit what level of complexity is shown on the screen depending on the scale of the window. The **VIEW LIMIT** command is one of three different commands which start with the keyword **VIEW**. **VIEW (ON | OFF)** is used in command files only and is covered below. The **VIEW** command is used to move or resize the view window.

The **AUTOPAN** command enables or disables the autopan feature. If autopan is enabled, whenever you move the cursor off of the screen, the view window will automatically shift in that direction. The **ARROW** command controls how the arrow keys can be used to manually pan the view window. The **ARROW** command also controls how the command history is traversed to repeat commands already executed. The **CURSOR** command defines the appearance of the cursor.

The cell you launch ICED™ to edit, is referred to as the root cell. Other cells can be edited without exiting from ICED™. This is especially useful when you need to edit cells nested in the root cell. The **EDIT** command can be used to edit a cell in what seems like a separate ICED™ session. **T_EDIT** (Transformed EDIT) is used to edit a child cell in the orientation used in the parent cell and using the

coordinate system of the parent cell. **P_EDIT** (EDIT in **P**lace) is similar to **T_EDIT**, except that the parent cell remains displayed on the screen while you edit the child cell.

See **Journaling
and Data
Recovery** in
Appendix B.

An edit session can be terminated with four different commands. The **EXIT** command will save changes. **LEAVE** will save changes (and overwrite the cell backup) only if the geometry in the cell has been modified. **QUIT** will not save any changes made. The **JOURNAL** command will not save changes either, but it will allow you to recover work through the use of the ICED™ journal file. The journal file records commands executed during the current session. It can be edited and then executed in a new ICED™ session.

The journal file is an example of a command file. A command file containing a list of ICED™ commands can be created with any ASCII text editor. These commands can then be executed as though you typed them all in at the keyboard. This feature of ICED™ allows you to create extremely sophisticated operations which can be executed with a single command.

Several commands control how command files are executed. The **@file_name** command executes a command file. The **LOG** command prevents some or all commands in the command file from being logged into the journal file. The **VIEW (ON | OFF)** command determines whether or not the display is continually refreshed during execution of a command file. The **\$comment** command displays a message on the screen and stores it in the log file. The **PAUSE** command is used to suspend the execution of a command file, usually until a key is pressed, allowing the user to see a message. The **XSELECT** command can disable the generation of embedded **SELECT** commands while the command file is processed. The **LIST** command saves a list of components to be acted upon. You should read about all of these options if you want to make serious use of command files.

There are several ways to export data from an ICED™ cell file. The **CIF** command creates CIF (Caltech Intermediate Form) files. The **STREAM** command is used to create Calma-GDSII Stream files. To export an intermediate file for rules checking, the **DRC** command can be used.

The **PLOT** command is used to create an intermediate file for plotting or bitmap export by the **MkPlot** utility.

The MkPlot utility is one of many ICED™-related utilities that must be executed in a separate console window. These utilities are fully described beginning on page 369. Two commands are used to execute console utilities. The **DOS** command pauses the layout editor while it executes a single command or opens a console window. The **SPAWN** command performs the same functions, but the editor will not be paused while the console operation executes.

Some miscellaneous commands are the **REDRAW** command which refreshes the screen, the **PACK** command which is used to report memory usage, and the **RULER** command that measures displacements. The **MENU** command is used to load customized menus. The **KEY** command assigns macros to keyboard function keys. The **TEMPLATE** command will display the ICED™ environment settings on the screen, or save them in a command file to create the same environment in future cells. Last, but not least, is the **UNDO** command that will reverse the effects of the last edit command.

Several other commands used exclusively in command files are documented only in the ICED™ Command File Programmer's Reference Manual.

See page 101
for an overview
of command
syntax.

The following pages list the ICED™ commands in alphabetical order. Each description includes examples that will help you use the more powerful features effectively. The syntax for each command is summarized briefly at the beginning of each command description.

@*file_name*

Execute a command file.

@*[path\]file_name*

or

@*

See an overview on command files on page 98.

Learn more about ICED_CMD_PATH on page 48.

Learn more about the AUXIL directory on page 44.

ICED™ accepts commands from the menus, the keyboard, or command files. The @*[path\]file_name* command causes ICED™ to execute the ICED™ commands in file *file_name*. The @* command executes the commands in the startup command file. (@* is covered below.)

If *file_name* does not include an extension, ICED™ will add an extension of .CMD to *file_name* before looking for the file. If you omit *path*, ICED™ looks for the file in the following directories in the order shown:

- the working directory,
- the directories specified by the ICED_CMD_PATH environment variable,
- and finally, the Q:\ICWIN¹⁵AUXIL directory.

Example:

@GEORGE

ICED™ will execute the commands in the file GEORGE.CMD if this file can be found in the working directory or in one of the other directories listed above.

Example:

@D:\WORK\NAND.LOG

ICED™ will execute the commands in the file D:\WORK\NAND.LOG.

¹⁵ Remember that Q:\ICWIN represents the drive and path where you have installed ICED™.

The *@** Command

This option is not intended for use in combination with the EXIT, LEAVE, or QUIT command line options.

These commands can be executed with menu options on the FILE menu.

Example:

This command causes ICED™ to execute the commands in the command file(s) specified on the ICED.EXE command line, the startup command file and/or the always command file. The startup command file is specified by the STARTUP parameter on the ICED™ command line in your project batch file. The always command file is specified with an ALWAYS command line option. The startup command file is executed automatically only when you create a new cell. It is used to initialize the layout editor environment (e.g. layer definitions). If the startup file has changed, and this is an old cell, you may want to execute the startup command file again with the *@** command to store the new settings in the cell. If an always command file was specified on the command line, it is executed after the startup command file automatically by the *@** command.

*@**

If this command is executed in the layout editor when the following command line was used to open the layout editor, then the NEW.CMD file would be executed. If there were no errors, the MYKEYS.CMD file would then be executed.

```
ICED MYCELL STARTUP=NEW ALWAYS=MYKEYS
```

The use of system macros is explained in the Command File

Programmers Reference. Also see the SHOW command.

If you want to execute only the startup command file, you can type the command file name explicitly with a normal *@file_name* command, or you can use the system macro that contains the file name of the startup command file. This means you do not have to remember the name of the file to execute it.

Example:

```
@%START.CMD  
@%ALWAYS.CMD
```

The first command will execute only the NEW.CMD command file. The second will execute only the MYKEYS.CMD file. The pair of commands is executed in this order by the single *@** command shown above.

The @%.cmd Menu Option

Command files with a "_" prefix are meant to be called from other files.

The ICED™ menu option '@%.cmd' allows you to select a command file to execute from lists of the command files in the working directory, each directory in the command file search path, and finally the AUXIL subdirectory. Only files with a .CMD extension and without a "_" prefix in the file name will be listed.

Commands Used Only in Command Files

Several commands are used to control the way ICED™ behaves while executing command files. The LOG command controls how commands are logged into the journal file. The VIEW command determines whether the screen is updated after every command in the command file. XSELECT is an important command when writing command files. It controls how commands that act on selected components behave when no components are selected.

Example:

**LOG SCREEN=OFF LEVEL=BRIEF
VIEW=OFF
XSELECT=OFF**

Using the three commands above at the start of a command file will:

- prevent ICED™ from echoing each command on the screen and prevent extra comments from being added to the journal file,
- turn off screen refresh until the command file is complete, and
- prevent commands acting on selected components from halting the command file if no components are selected.

The first two commands will make the command file execute more quickly.

Learn more about command files in the **Classroom Tutorials Manual**.

All three of the commands above are described in this manual. There are many more commands intended only for command files described in the Command File Programmer's Reference Manual. These include macro (variable) definition and assignment, conditional statements, and component or cell iteration commands.

Nested Command Files

The VIEW, LOG, and XSELECT modes selected by using those commands in a command file will be used in any *@file_name* commands nested within the file. The VIEW and XSELECT modes can be overridden with commands in the nested *@file_name* command files.

If a command file contains a LOG=OFF command, logging to the journal file remains off until the command file is completed. There is no way to override the LOG OFF mode.

Command files can be nested up to 16 deep.

Command File Errors

Any error encountered during execution of a command file will cause ICED™ to stop processing the remainder of the file. If an error is encountered during execution of a nested command file, not only will the rest of the nested command file be ignored, but rest of the command file that contained it will be ignored as well.

Details on the journal file are covered on page 99.

If an error is encountered during execution of a command file, the error message is shown on the ICED™ screen. The error is also logged into the journal file.

Once a command file is executed, only the last command in the file can be reversed by the UNDO command. If necessary, you can use the JOURNAL command and a modified journal file to recover the cell as it was before the command file was executed. (See Appendix B.)

See details about the three different types of command file comments on page 121.

One trick to ensure that your command file has run to completion is to add the command \$SUCCESS at the end of each command file. A *\$comment* command at the end of a command file causes an echo on the ICED™ screen that will remain there after the command file is complete. Using this method makes it obvious that the command file has completed successfully.

Order of Execution

If the *@file_name* command is issued with other commands on the same command line using semicolons to separate them, the other commands will execute prior to the commands in the command file.

Example:

@CMDFILE;VIEW OFF; XSELECT OFF

Refer to the **Command File Programmers Reference Manual** for a complete description of command files.

In this example, the command @CMDFILE is parsed first by the command interpreter. Then the VIEW OFF command is executed as though it is the first line of the command file. The XSELECT OFF command is executed next. Finally, the commands in the file CMDFILE.CMD are executed.

\$comment

Add comment to journal file and screen.

\$comment

or

\$\$comment

See page 99 to learn more about journaling.

Whenever ICED™ processes a command with a '\$' as the first character, the remainder of the line is treated as a comment. The '\$' and the remainder of the line will be logged into the journal file as a comment and the line will echo on the screen. Even if there is a valid command in the comment, it will not be executed.

These types of comments are useful primarily in command files.

Example:

\$My command file completed successfully; VIEW IN 3

The entire line above will be treated as a comment and be will copied into the journal file and displayed on the bottom of the editor window exactly as shown. The VIEW command will **not** be executed.

Although commands in \$comments will not be executed, macro substitution and expression evaluation will be performed.

Example:

**LOCAL #COUNTER=1
\$ COUNTER = %COUNTER**

Learn about macros in the **Command File Programmer's Reference Manual**.

The comment will produce the following echo on the screen and in the journal file:

\$ COUNTER = 1

Comment commands can be used to mark locations in the journal file. Let us say that you are performing a series of complicated editing tasks. After each

See Recovering
From Mistakes
in Appendix B.

successful step, you can add a comment that the step is complete. If the last step you performed went badly, and you need to recover the state of the design several steps back to correct the problem, you can edit the journal file using the comments to locate the step where things went wrong. After the journal file is edited, you can use it to recover your work up to the step where problems began.

Differences Between \$comments, \$\$comments and !comments

During long command files, you may want to turn the logging of commands to the command file off with the LOG command. This will speed up a long command file considerably. This also turns off the display and logging of \$comments.

Adding a
PAUSE 0
command after
a *\$comment* will
leave the
comment on the
screen until a
key or mouse
button is
pressed.

\$\$comments will still be displayed and logged to the journal file even when the LOG command prevents the display and logging of other commands. This is the only difference between \$comments and \$\$comments.

!comments can also be used to create comments in command files, but these comments will not be logged into the journal file or displayed on the screen. They are used only as documentation within the command file itself.

ADD***Add a component to the drawing.***

ADD BOX [LAYER=*layer_id*] [[AT] *pos1 pos2*]
ADD POLYGON [LAYER=*layer_id*] [[AT] *pos_list*]
ADD CIRCLE [LAYER=*layer_id*] RADIUS=*radius* [N_SIDES=*n*] [[AT] *pos*]
ADD RING [LAYER=*layer_id*] RADII=*rad0, rad1* [N_SIDES=*n0, n1*] [[AT] *pos*]
ADD WIRE [LAYER=*layer_id*] [WIDTH=*width*] [TYPE=(0 | 2)] [[AT] *pos_list*]
ADD ARC [LAYER=*layer_id*] RADIUS=*radius* [WIDTH=*width*] ...
... [TYPE=(0 | 2)] [N_SIDES=*n*] ANGLES=*ang0, ang1* [[AT] *pos*]
ADD SECTOR [LAYER=*layer_id*] RADIUS=*radius* [N_SIDES=*n*] ...
... ANGLES=*ang0, ang1* [[AT] *pos*]
ADD LINE [LAYER=*layer_id*] [[AT] *pos_list*]
ADD TEXT "*string*" ["*string_2*"... "*string_n*"] [LAYER=*layer_id*] [SIZE=*size*] ...
... [MX | MY] [HORIZONTAL | VERTICAL | R=*rot_code*] [JUST=*just_code*] ...
... [[AT] *pos*]
ADD CELL=*cell_name* [MX | MY] [R=*rot_code*] [[AT] *pos*]
ADD ARRAY=*cell_name* N=*nx,ny* [STEP=*disp_x, disp_y*] [MX | MY] ...
... [R=*rot_code*] [[AT] *pos*]

In addition, there are two optional parameters which can be used in any of the ADD commands. [OFFSET=*off_x, off_y*] can be used before the AT keyword. OFFSET is only useful in command files. The [ID=*idint*] parameter is ignored when the ADD command is executed. This feature is useful when executing files generated by the SHOW command.

The ADD command is used to add components to a drawing. ICED™ supports seven basic component types: boxes, polygons, lines, wires, cells, arrays, and text. The ADD CIRCLE, ADD SECTOR, and ADD RING commands actually add polygons to approximate circles, sectors, and rings. The ADD ARC command adds a segmented wire to approximate an arc.

Default values for optional parameters can be set. See the table below to find the correct command to set each parameter's default value. ADD commands selected from the ICED™ menus usually use these default values.

Parameter	Purpose	Components using this parameter	Command used to set default
ANGLES	Sets initial and final angles	ARC and SECTOR	none
[AT] <i>pos</i> or <i>pos_list</i>	Defines vertex or center coordinates ¹⁶	All	none
<u>HORIZONTAL</u> or VERTICAL	Sets text orientation	TEXT	none
ID	Ignored (sometimes set by SHOW command)	All	none
JUST	Sets text justification	TEXT	TEXT or USE
LAYER	Defines which layer component will be on ¹⁷	All except CELL and ARRAY	LAYER or USE
MX MY	Used to mirror component	TEXT, CELL and ARRAY	none
N_SIDES	Sets number of sides used to approximate circle	CIRCLE, RING, ARC, and SECTOR	USE
OFFSET	Offset coordinates ¹⁸	All	none
R	Used to rotate component	TEXT, CELL, and ARRAY	none
RADIUS or RADII	Defines radius of circular component	CIRCLE, RING, ARC, and SECTOR	none
SIZE	Defines text size	TEXT	LAYER
STEP	Sets distance between cells in an array	ARRAY	none
TYPE	Sets type of wire ends	WIRE and ARC	USE
WIDTH	Determines wire width	WIRE and ARC	LAYER

Figure 11 ADD Command Parameters at a glance.

¹⁶ See Entering ADD Command Positions on page 125 for an explanation of entering coordinates.

¹⁷ See The [LAYER=layer_id] Parameter on page 124 for details.

¹⁸ Refer to The OFFSET Parameter on page 127 for more information.

Keyword Order

ICED™ places the following restrictions on the order in which keywords appear in the ADD command:

- The component type keyword (BOX, POLYGON, etc.) must be the first keyword.
- If it appears, the LAYER keyword must be the second keyword.
- If the STEP keyword appears in the ADD ARRAY command, it must come after the N keyword.
- If it appears, the OFFSET keyword must come directly before the position list (or AT keyword if it is used). OFFSET is valid only if the *pos* or *pos_list* parameter is supplied with the command. (See The OFFSET Parameter on page 127 for more details on OFFSET.)
- If it appears, the AT keyword must be the last keyword. In any event, the position list (if any) must be at the end of the command.

The [LAYER=*layer_id*] Parameter

All ADD commands (except for ADD CELL and ADD ARRAY) add components to a specific layer. You can select this layer with the *layer_id* parameter. If no *layer_id* parameter is supplied in the command, the default layer will be used. This default layer is set with the LAYER or USE LAYER commands. You can execute USE with no other parameters to have ICED™ report the current default layer.

Layer number 99 has special uses for the DRC program. If you use this program, avoid adding components to layer 99.

If you wish to use the LAYER keyword in the ADD command, *layer_id* can be a layer name defined by the LAYER command or a layer number. (In either case, it is the layer number that is stored with the component definition.) Layers 250:255 should be avoided when using the ADD command to create maskable components. These layers are used by IC Editors, Inc. and other command file developers for scratch work.

Entering ADD Command Positions

All ADD commands need at least one coordinate pair of real numbers to supply a position for placing the component in the drawing. Positions can be supplied in the command, or digitized with the mouse.

When positions are included in a command, they must be the last parameters in the command. The AT keyword is not required to enter the position parameters. However, using AT does make the commands more readable. Parentheses and commas can be used to make commands more readable, but they are also optional.

Example: **ADD BOX AT (20.5 , 0) (-100.5 , 10)**
 ADD BOX 20.5 0 -100.5 10

These two ADD commands are equivalent.

When you do not supply the required position(s) in the command, ICED™ expects you to digitize them with the mouse. Move the cursor to the desired position (ICED™ will report the exact cursor coordinates on the bottom of the screen). Then click the left mouse button once to digitize each position. If several positions are required, you can often indicate the last position has been entered by clicking the right mouse button **after** digitizing the final position.

See the
RESOLUTION
 and **SNAP**
 commands for
 details on using
 the snap grid.

Coordinates entered with the mouse will snap to the grid set by the SNAP command. Coordinates typed in the ADD command will be stored rounded to the nearest database unit. The database unit is the smallest unit of measure that ICED™ can use when storing coordinates.

See the **NDIV**
 command line
 parameter for
 more
 information on
 how coordinates
 are stored.

$$1 \text{ database unit} = 1 \text{ user unit} / \text{NDIV}$$

where NDIV represents the number of divisions in a user unit. By default, NDIV=1000. NDIV can be overridden on the command line when ICED™ is launched.

Using Nested View Commands while Digitizing Positions

When you digitize positions during an ADD command (or any command that requires digitized positions), you can press the <Esc> key (or the middle button of a three button mouse) and use the nested view menu to change the view window during the command.

Learn about automatic panning of the view window in the description of the **AUTOPAN** command.

See the **SPACER** command to learn about a visual aid used to digitize coordinates at least a minimum distance away from existing components.

Let us say that you are digitizing vertices during an ADD WIRE command. The next vertex you need to digitize is far beyond the left side of current view window. You could move the cursor off of the left side of the window and keep panning over to the desired view window. Another way is to use the nested view menu with steps similar to those shown below.

Press the <Esc> key to display the nested view menu.

Zoom out with the menu option OUT%->5.0. The nested view menu disappears.

Press <Esc> again and the nested view menu reappears.

Zoom in around the general area of the next vertex by using the BOX option of the nested view menu and digitizing the corners of the new view window.

The nested view menu disappears again after zooming in. The ADD WIRE command is still executing. Now digitize the desired vertex with the left mouse button.

You can continue to change the view window while you digitize additional vertices of the wire component. After the ADD WIRE command is completed, you can use the VIEW LAST command to restore the original window. During a command, you can use the HOME option on the nested view menu to restore the view window in place at the beginning of the ADD command.

The OFFSET Parameter (only used in command files)

See the
@file_name
command to
learn about
command files.

When ADD commands are typed in a command file, there is an option to add an offset to each coordinate position supplied with the command. This can make repetitive ADD commands much easier to write. The OFFSET parameter is only valid when the coordinate positions are supplied with the command.

If you use the offset feature, the **OFFSET=off_x, off_y** parameter must be the next to last parameter in the command, directly before the position parameter(s). The values of *off_x* and *off_y* must be real numbers. The value of *off_x* will be added to each x-coordinate of the position list. The value of *off_y* will be added to each y-coordinate of the position list. ICED™ will report an error if no positions are supplied in the command after the OFFSET parameter.

Example: **ADD WIRE WIDTH=5 OFFSET=10,20 AT (0,0) (5,0) (5,10)**

In this example, the wire center line would be defined by the points (10,20) (15,20) and (15,30).

Vertex Calculations

When you add circles, rings, arcs, or sectors, vertices must be calculated by ICED™. The result of these calculations depends on the current settings of the resolution mode and resolution grid step size (set with the RESOLUTION command) and the NDIV parameter (see page 125.)

A database unit
is defined above
in **Entering
ADD
Command
Positions** on
page 125.

If the resolution mode=HARD, the vertex points calculated by ICED™ will be rounded to lie on the resolution grid. When the resolution mode=SOFT, points are not forced to lie on the resolution grid, but are rounded to the nearest database unit.

Most mask processing software will allow vertex points to lie off of the resolution grid and will perform any required approximations. However, if your software does not support off-grid vertices, or if you prefer to avoid having changes made to your design at the mask-making step, set the resolution mode to HARD to create your data on grid.

When ICED™ rounds coordinate data for a component with many vertices, several vertices may be rounded to the same point or several points may be rounded to be collinear. In this case, ICED™ will issue an error message similar to "Radius too small -- change N_SIDES ?" and no component will be created.

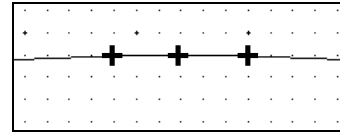


Figure 12

Example:

RESOLUTION MODE=HARD STEP=.25
ADD CIRCLE RADIUS=3 N_SIDES=128

This ADD command will fail to create the requested circle when the resolution mode is HARD. If the mode is SOFT, the command is more likely to succeed, but your mask processing software may perform a large number of approximations on those 128 vertex points, significantly distorting the shape of the circle.

More details on circular component angles and vertices is provided in ADD ARC on page 139.

A value of N_SIDES=32 is usually sufficient for circles, arcs, rings, and sectors. The maximum spacing between a circle and its polygon approximation is given by:

$$r - r \cos(\frac{1}{2}\phi) \quad \text{where } \phi = \frac{2\pi}{N_SIDES}$$

or

$$2 r \sin^2 \left(\frac{\pi}{2 N_SIDES} \right)$$

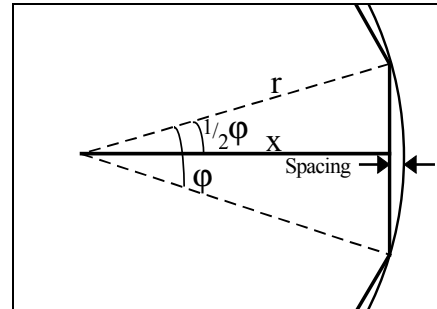


Figure 13

If N_SIDES=32, the maximum spacing is approximately $r/200$.

ADD BOX [LAYER=*layer_id*] [[AT] *pos1 pos2*]

See page 124 for details on using the LAYER parameter. Entering coordinates is explained on page 125.

ADD BOX will add a rectangle to the drawing. The sides will be vertical and horizontal.

The *pos1* and *pos2* parameters define the coordinates of opposite corners of the box. It does not matter which pair of opposite corners is defined or the order in which the corners are listed.

Example:

ADD BOX LAYER=POLY AT (120.5, -34) (130.5, -30)
ADD BOX LAYER=POLY AT (120.5, -30) (130.5, -34)

Either command will add a rectangle on layer POLY with corners at (120.5, -34) (120.5, -30) (130.5, -30) and (130.5, -34)

ADD POLYGON [LAYER=*layer_id*] [[AT] *pos_list*]

See page 124 for details on using the LAYER parameter. Entering coordinates is explained on page 125.

ADD POLYGON will add a polygon to the drawing. The maximum number of sides is 199. The angle between sides digitized with the mouse can be constrained to multiples of 45° or 90° by the SNAP command. When the snap angle is set to 0°, or when coordinates are typed in the command, any angle between sides is allowed.

The *pos_list* parameters define the coordinates of the vertices. At least 3 vertices must be defined. If typed in the command, *pos_list* must be a list of coordinate pairs of real numbers. If you define the vertices of the polygon with the mouse, the recommended method to close the polygon is to redigitize the first vertex.

Example:

ADD POLY LAYER=M1 1,1 10,1 2,2

This command will add a polygon on layer M1 with corners at the coordinates indicated. Note that POLYGON is usually abbreviated to POLY. Note that these points define a triangle with angles which are not multiples of 45°. (If you digitized these points with the mouse instead of typing them in the command, ICED™ would issue an error message unless the current snap angle (set by the SNAP command) was set to 0, indicating that any angle is acceptable.)

The command above would create a closed shape despite the fact that the last vertex is not a repeat of the first. The polygon is closed by connecting the last vertex provided to the first vertex. If the final vertex is a repeat of the first, it is ignored.

ICED™ will optimize the polygon before saving it. Redundant points and sections with zero area will be removed from the polygon definition.

Example:

ADD POLY (0,0) (6,0) (3,5) (3,10) (3,5)

See the **SPACER** command to learn about a visual aid used to digitize coordinates at least a minimum distance away from existing components.

The current default layer is always reported in the command prompt below the view window

Despite the fact that this example defines 5 vertices, it will create a triangle. The 5 vertices chosen are shown in Figure 14. The triangle which ICED™ will create from these vertices is shown in Figure 15. Note that the vertex at (3,10) is between 2 sides which are on top of each other. When this is the case, the ADD POLY command erases the part of the polygon which has zero area, leaving a triangle. The second vertex at (3,5) is now a repeat of the previous vertex, so it too is ignored. Since no LAYER keyword is present, ICED™ will create the polygon on the default layer.

ICED™ will also remove points of a polygon that are redundant. If a polygon is defined with several points in a straight line, it will remove the points which are in the middle. Removing these points results in exactly the same geometry, but requires less space to store the data, and speeds up any processing on this component.

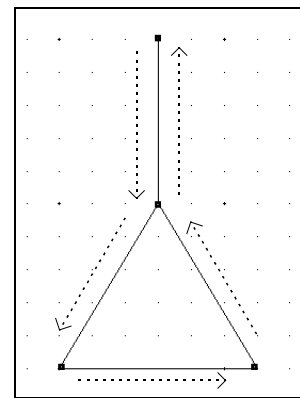


Figure 14: Points selected for the ADD POLY command.

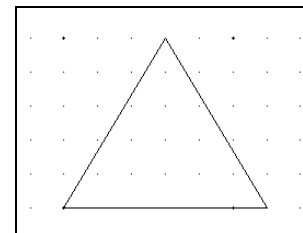


Figure 15: Polygon created by the points in Figure 14.

Example: **ADD POLY (0,0) (1,0) (2,0) (3,0) (4,0) (5,0) (5,2) (0,2)**

This example defines several connected points that lie on a straight line. The points in the middle of this line are unnecessary to define the rectangle. See Figure 16. ICED™ will ignore the redundant points (1,0), (2,0), (3,0), and (4,0). Since the remaining points define a simple rectangle, ICED™ will instead create the box component shown in Figure 17. The box component defines exactly the same geometry and is simpler to process and store.

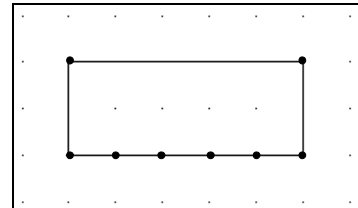


Figure 16: Redundant points used to define polygon.

Example: **ADD POLY**

You can back up (undigitize) points while digitizing a polygon or wire.

Since no *pos_list* is provided with this ADD command, you must define the vertices of the polygon with the mouse. The recommended way to close the polygon, after defining the final vertex, is to redigitize the first vertex. Using this method, you can see exactly how the polygon is closed as you reselect the first vertex. The other method to close the polygon is to click the right mouse button **after** digitizing the last vertex. In this case, some assumptions are made about how you want the polygon closed.



Figure 17: Box created from the points in Figure 16.

Use the **SNAP** command to set the snap angle

Consider Figure 18 assuming that the snap angle is set to 90° or 45°. When the snap angle is non-zero, ICED™ will extend the existing sides vertically and horizontally when you click the right mouse button. Since ICED™ can extend the two sides to intersect at a point on the resolution grid, it will assume that you meant to create a box if you click the right mouse button. If the snap angle is set to 0°, the polygon would be closed with a slanted line segment which directly connects the first and last vertices.

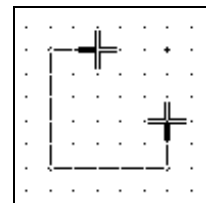


Figure 18: This polygon can be closed.

Now consider the polygon shown in Figure 19. If the snap angle is non-zero, ICED™ cannot close the polygon by extending existing sides and nothing will happen if you click the right mouse button after digitizing the last vertex. However, if the snap angle is set to 0°, the polygon will be closed with a slanted line segment. If the upper right hand corner was digitized in error, and should have had the same y-value as the upper left corner, you may never realize that the point was not digitized correctly.

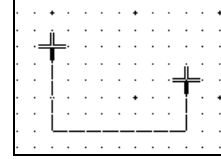


Figure 19: This polygon may not be closed automatically.

When digitizing maskable components, it is strongly recommended to close polygons by redigitizing the first vertex instead of using the right mouse button. It is much more obvious when points are misdigitized. Problems like the one described above are easily avoided.

Example: **ADD POLY AT (0, 0) (.1, 0) (0, .1)**

See the **@file_name** command to learn about command files.

ICED™ will not let you create a polygon of nearly zero area. If you typed the above command in ICED™, you would get a warning message and no component would be created. If the command is in a command file, ICED™ assumes that you want to continue the command file despite the warning and would continue to process the other commands in the command file.

ADD CIRCLE [LAYER=layer_id] RADIUS=radius [N_SIDES=n] [[AT] pos]

ADD CIRCLE will add a polygon approximating a circle to the drawing. The radius of the circle is given by *radius*. *radius* must be a positive real number. This parameter is **not** optional. The radius is measured from the center point to the center of a side, **not** from the center point to a vertex. See Figure 20.

Refer to **Vertex Calculations** on page 127 for details on vertex placement.

The N_SIDES keyword defines the number of sides of the polygon used to approximate the circle. It must be an integer in the range 3:199. If N_SIDES is omitted, the default provided by the USE command will be used. The most commonly used value for N_SIDES is 8. If you need a more circular component, a value of 32 for N_SIDES is almost always sufficient. Higher values often cause too many approximations when post-processing software rounds vertex data. The angle between connected sides is **not** constrained by the snap angle.

Defining positions is explained on page 125.

Example:

The *pos* parameter is used to define the location of the center of the circle.

ADD CIRCLE R=5 N_SIDES=8 AT (15, 15)

See page 124 for details on the default layer and the LAYER parameter.

This command will add the octagon shown in Figure 20 on the default layer with a radius of 5 user units. Note that the RADIUS keyword can be abbreviated to R because no other keyword of the ADD CIRCLE command starts with the letter R. The center of the polygon will be at the point (15,15). ICED™ always draws the first side of a polygon or circular component on the bottom, parallel to the x-axis.

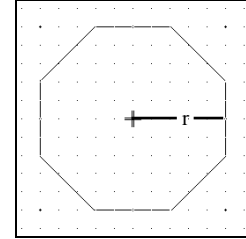


Figure 20:
Octagon created with ADD CIRCLE.

ADD RING [LAYER=*layer_id*] RADII=*rad0*, *rad1* [N_SIDES=*n0*, *n1*] [[AT] *pos*]

See page 124 for details on using the LAYER parameter.

ADD RING will add a polygon approximating a circle with a hole in it. The radius of the inner circle is given by *rad0*. The radius of the outer circle is given by *rad1*. Both parameters must be positive real numbers. *rad1* must be greater than *rad0*. These parameters are **not** optional.

The N_SIDES keyword defines the number of sides for the inner and outer circle approximations. *n0* defines the number of sides for the inner circle. *n1* determines the outer circle. *n0* must be an integer in the range 3:96. *n0* + *n1* cannot exceed 195. *n1* can be less than *n0*, as shown in Figure 22. If N_SIDES is omitted, the default provided by the USE command will be used for both *n0* and *n1*. The angle between successive sides is **not** constrained by the snap angle.

Entering coordinates is explained on page 125.

The *pos* parameter defines the coordinates at the center of the ring.

Example: **ADD RING R=3,5 N_SIDES=10,12**

Refer to **Vertex Calculations** on page 127 for details on vertex placement for all circular components.

The command above will add the polygon shown in Figure 21. The LAYER parameter is not provided, so the layer will be determined by the current value of the default layer. The radius of the inner circle is 3 user units and it has 10 sides. The radius of the outer circle is 5 user units and it has 12 sides. The center position is not provided in this example, so it must be defined with the mouse.

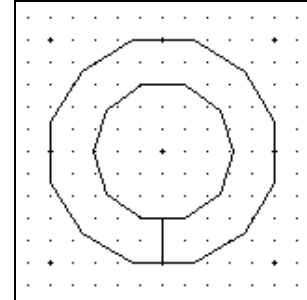


Figure 21: Polygon created with ADD RING.

Example: **ADD RING R=3,5 N_SIDES=16,4**

This example demonstrates how *n1*, the number of sides of the outer circle, can be less than *n0*, the number of sides of the inner circle.

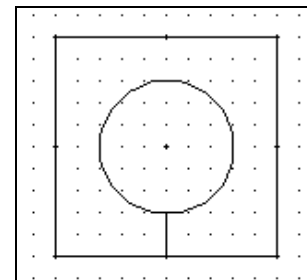


Figure 22: RING component with N_SIDES=16,4.

**ADD WIRE [LAYER=layer_id] [WIDTH=width] ...
... [TYPE=(0 | 2)] [[AT] pos_list]**

See page 124 for details on using the LAYER parameter.

ADD WIRE will add a wire to the drawing. In ICED™, the backbone of a wire is a centerline formed from a connected series of points. The wire expands one half of the *width* parameter on either side of this centerline.

The **LAYER** command associates a default wire width with every layer.

The **WIDTH** parameter determines the width of the wire from edge to edge. *width* must be a positive real number. If it is omitted, the default width assigned to the wire's layer by the LAYER command will be used.

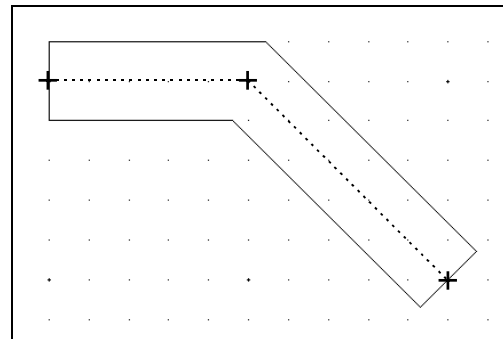


Figure 23: The line connecting the wire vertices is dotted. The wire outline is the solid line. This is a type 0 wire.

A database unit is explained in more detail on page 125.

It is important to know that many calculations involving wires use one half of *width*. If $width/2$ is not a whole number of database units, ICED™ will display an error message, and the ADD WIRE command will fail. A database unit is the smallest unit of measurement ICED™ can store. If the database unit is defined as .001, then a wire width of 1.001 is invalid since ICED™ cannot store a value of 0.5005.

See the **SPACER** command to learn about a visual aid used to digitize coordinates at least a minimum distance away from existing components.

The TYPE parameter controls the method used to define the ends of the wire. ICED™ supports wire types 0 and 2 which correspond to Calma path types 0 and 2. Type 0 wires have flush ends (the end of the wire is flat against the end point). See Figure 23 for an example. Calma type 1 wires, using rounded ends, were intended for printed circuit board layout and are **not** supported by ICED™. Type 2 wires extend one wire half-width past the digitized points on the ends of the wire. See Figure 24. The default for the TYPE parameter is set by the USE command.

Entering point data is explained in more detail on page 125.

The values in *pos_list* define the points on the center line. The maximum number of points is 200. If you define several points in a line, ICED™ will remove any redundant points and save the least number of coordinates required to define the wire. Doubled over sections will be removed. If you omit *pos_list* and define the points with the mouse, press the right button **after** you digitize the final point to indicate that you are finished.

Example:

**ADD WIRE LAYER=M1 WIDTH=2 ...
... TYPE=2 AT (0,0) (5,5) (5,10)**

This command will add the wire component shown in Figure 24. The wire will be on the M1 layer. The width of the wire is 2 user units. Since it is a type 2 wire, the ends of the wire extend 1 user unit (half of the *width* parameter) past the points on the end of the wire.

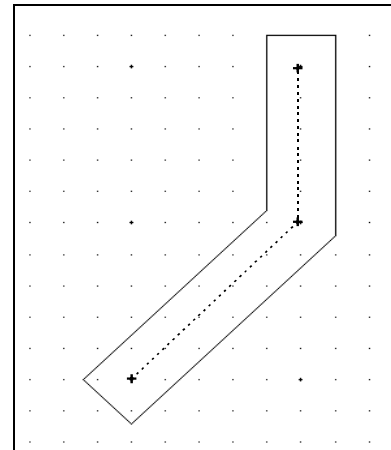


Figure 24: A wire created by ADD WIRE with the center-line indicated by a dotted line. This is a type 2 wire.

The **SNAP** command defines the snap angle.

You can back up (undigitize) points while digitizing a polygon or wire.

You can use the DRC program available separately from IC Editors to find self-intersecting shapes.

The above example would fail if the current snap grid is set with a snap angle of 90°. The snap angle controls what angles are valid between points on the wire. The snap angle limitation applies whether the points on the wire centerline are defined with the mouse or provided with the ADD WIRE command.

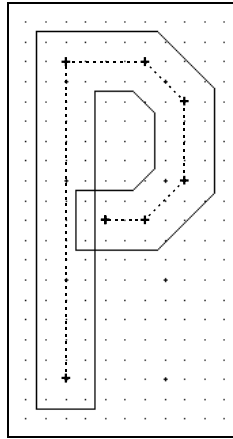


Figure 25: Self-intersecting type 2 wire used to create a letter.

Care must be used when creating wires. Many types of post-processing software will translate wires to polygons. If a wire is self-intersecting, then errors will occur as it is translated to a self-intersecting polygon. The self-intersecting wire shown in Figure 25 is likely to cause problems.

The calculations used to determine the wire outline vary from one software package to another, and the wire outline drawn by ICED™ may not be the same as one generated by your post-processing software. Avoid sharp angles and short end segments (i.e. less than the half-width of the wire). Small jogs less than the wire width are acceptable as long they are not on the end segments of the wire. Any wire for which ICED™ draws an unexpected outline will almost certainly cause problems when you export the data to another application.

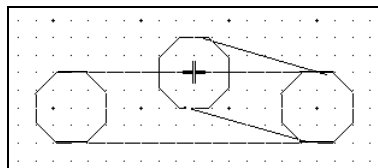


Figure 26: Digitizing the coordinates of a type 0 wire with an acute angle.

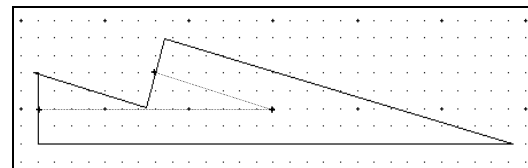


Figure 27: This undesirable wire outline is generated from the dotted center line created in Figure 26.

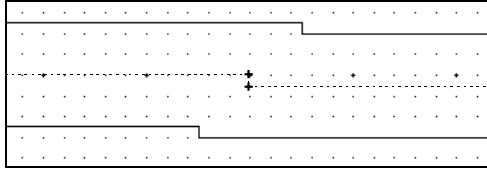


Figure 28: Small jogs less than $width/2$ are acceptable in middle sides only.

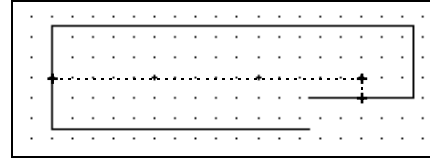


Figure 29: End sides less than $width/2$ will produce odd outlines on a type 0 wire.

ADD ARC [LAYER=*layer_id*] RADIUS=*radius* [WIDTH=*width*] ...
... [TYPE=(0 | 2)] [N_SIDES=*n*] ANGLES=*ang0*, *ang1* [[AT] *pos*]

See page 124
for details on
using the
LAYER
parameter.

ADD ARC will add a wire approximating an arc to the drawing. The radius of the arc is given by *radius*. *radius* must be a positive real number. This parameter is **not** optional. The radius is measured from the center point to the center of a side of the center line of the wire, **not** from the center point to a vertex. See Figure 30.

The WIDTH parameter determines the width of the arc wire from side edge to side edge. *width* must be a positive real number. If it is omitted, the default value associated with the layer will be used. This default value is set by the LAYER command.

The TYPE parameter controls the type of wire used to approximate the arc. Type 0 wires have flush ends (the end of the arc is flat against the end vertex). Type 2 wires extend past the computed points on the ends of the arc center line by half of *width*. The default for TYPE is set by the USE command.

The N_SIDES keyword defines the number of sides of the circle the arc is a segment of. It must be an integer in the range 3:197. If this parameter is not provided, the default used is the N_SIDES parameter set by the USE command. More details on this parameter follow in the second example. The angle between successive sides is **not** constrained by the snap angle.

The ANGLES keyword is used to define the initial and final angles of the arc. 0° represents 12 o'clock, and each angle is measured proceeding clockwise. *ang0* and *ang1* must be real numbers in the range -720.0 : 720.0. The angles are normalized to lie in the range 0° : 360°. The start angle, *ang0* will be measured clockwise from 12 o'clock, then the arc will proceed clockwise to *ang1*.

Entering point data is explained in more detail on page 125.

Example:

The *pos* parameter defines the center of the arc. *pos* must be a coordinate pair of real numbers. If *pos* is omitted, you must define it with the mouse.

LAYER M1 WIDTH=2.0
ADD ARC LAYER=M1 R=10 TYPE=2 N_SIDES=16 ANGLES=0, 135

This command will add the wire shown in Figure 30. The wire will be created on layer M1. Since no width parameter is supplied with the command, the default width of 2 user units assigned to the M1 layer by the LAYER command is used.

The radius of the arc is 10 user units. Since it is a type 2 wire, the wire ends extend past the vertices calculated by ICED™.

Note that the N_SIDES parameter defined the number of sides the arc would have if it was a full circle, not the number of sides of the created arc.

Since no center position is supplied with the above example, the mouse is used to place the arc in the drawing. Note that the cursor used to place the component is on the center point of the arc, not on the geometry of the arc wire itself.

Example:

ADD ARC R=10 N_SIDES=18 ...
...TYPE=0 ANGLES=-90 , 370

In this example, the type 0 wire shown in Figure 31 is created. Note that the end of the wire at -90° is even with the center point of the arc. If it was a type 2 wire, it would extend past that point.

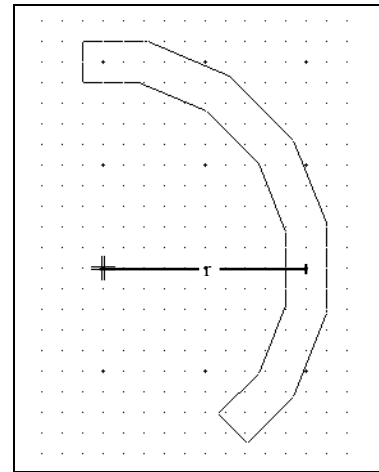


Figure 30: A wire created by the ADD ARC command.

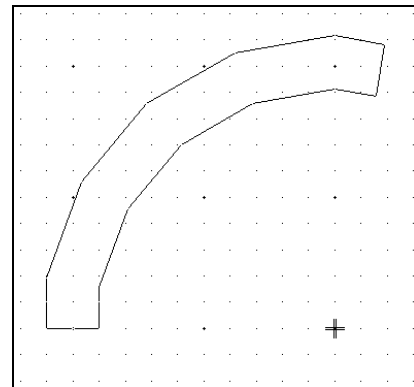


Figure 31: A type 0 wire created by ADD ARC.

The initial angle is normalized to 270°. The arc then sweeps to the final angle, normalized to 10°. Because no LAYER parameter is provided, the wire will be on the current default layer.

The N_SIDES=18 parameter represents the number of sides the arc would have if it was a full circle. The following discussion describes how the number of sides of the arc and the angles between the sides are determined by ICED™. If this does not concern you, feel free to skip ahead to ADD SECTOR on page 140.

Refer to **Vertex Calculations** on page 127 for details on vertex placement for all circular components.

First let us consider the entire circle and the angles between sides. N_SIDES represents the number of sides of the circle approximation and also the number of 'bends' or angles in the wire. A circle has a total of 360°. Therefore, the angle ϕ between successive sides is given by:

$$\phi = \frac{360^\circ}{N_SIDES}$$

If N_SIDES=18, $\phi=20^\circ$.

Now let us consider arcs. n, the number of 'bends' or angles in the arc, depends on the angle of the arc as well as N_SIDES. The angle θ of the arc is just the difference between the initial and final angles. ϕ , the ratio of θ to n, must be the same as 360° to N_SIDES.

$$\phi = \frac{\theta}{n} = \frac{360^\circ}{N_SIDES}$$

Looked at another way:

$$n = \frac{N_SIDES * \theta}{360^\circ}$$

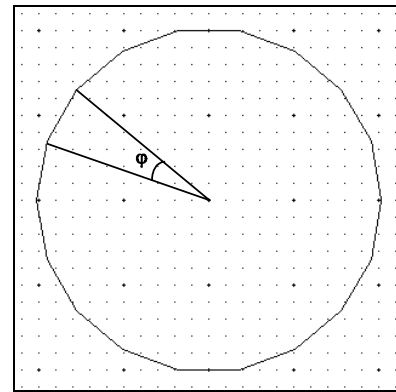


Figure 32: A circle approximation with 18 sides.

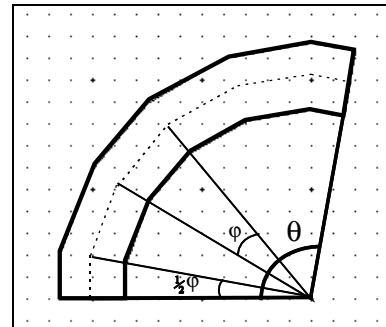


Figure 33: An ARC approximation with N_SIDES=18.

The number of bends must be an integer, so the calculated value of n is rounded to the nearest integer. To make the arc wire more circular, one side is split into two equal shorter sides on the ends. Therefore, the angles on the ends are one half of ϕ .

In the example above, $\theta=100^\circ$ and $N_SIDES=18$, so $n=5$. There are 5 bends in the arc, and 4 whole sides and 2 halves, making a total of 5 sides. The angle ϕ of the middle segments of the arc is $100^\circ/5=20^\circ$. The two angles on the ends are one half of ϕ or 10° .

ADD SECTOR [LAYER=*layer_id*] RADIUS=*radius* [N_SIDES=*n*] ...

... ANGLES=*ang0, ang1* [[AT] *pos*]

ADD SECTOR will add a polygon approximating a sector of a circle to the drawing. The parameters are similar to the ADD ARC command with the exception that the WIDTH and TYPE parameters are absent. The WIDTH and TYPE parameters apply only to wires.

Example: **ADD SECTOR R=10 ANGLES= -90,370**

This ADD SECTOR command uses the same parameters as the ADD ARC example above (except that the non-applicable WIDTH parameter is absent). The polygon shown in Figure 34 is created.

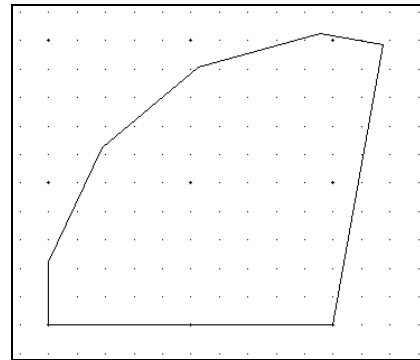


Figure 34: Polygon created with ADD SECTOR.

ADD LINE [LAYER=*layer_id*] [[AT] *pos_list*]

See page 124 for details on using the LAYER parameter.

ADD LINE will create a line in the drawing. Lines are used only to document your drawing. They do **not** produce maskable geometry. A line is always drawn with a width of 1 pixel regardless of the display scale.

See the **FILL** command to turn fill on and off.

The color of lines and text components varies depending on whether or not fill is turned on. If fill is on, lines and text are drawn in white regardless of the layer they are on. If fill is off, then lines and text are drawn in the color of the layer. This prevents text and lines from being hidden by filled components they were meant to label.

Entering coordinate data is explained in more detail on page 125.

The *pos_list* is composed of coordinate pairs of real numbers which define the vertices of the line. If you define them with the mouse, press the right button after you digitize the last point of your line. The snap angle restricts the points you can digitize with the mouse.

**ADD TEXT "string" ["string_2"... "string_n"] [LAYER=layer_id] [SIZE=size] ...
... [MX | MY] [HORIZONTAL | VERTICAL | R=rot_code] [JUST=just_code] ...
... [[AT] pos]**

See page 124 for details on using the LAYER parameter.

ADD TEXT will create a text component in the drawing. Text is used to document your drawing only. Text components do not translate to maskable geometry.

The NLE program can use text components to label nets or devices.

The quotes are **not** optional when you provide the string in the ADD TEXT command. There are four different quote characters available in ICED™; ", ', ~, and `. If *string* contains one of these characters, a different quote character must be used to surround it. For example, if the command string contains a double quote (") you can surround it with single quotes ('). There is no way to create a text component containing all four quote characters.

You can add more than 1 line of text at a time by using more than one quoted string. Each line should be delimited by quotes. There is no upper limit on the number of lines, but the total number of characters in all lines (including the newline characters at the end of each line) cannot be greater than 512.

Example:

**ADD TEXT "This is my first line of text" "This is my second line" ...
..."This line uses 'quotes'" AT 0,0**

In the example above, the double quotes delimit the text strings which will be added to the drawing. The single quotes will appear in the text component just

as they are. Since single quotes are used in the string, a different quote character must be used to delimit it.

The **TEXT** command controls many defaults for **ADD TEXT** commands.

Note that lower case characters are present in the strings above. How these lower case characters are treated depends on the current setting of the **LOWER_CASE** parameter set by the **TEXT** command. If **LOWER_CASE** is disabled, the lower case characters will be forced to upper case. Some post-processing programs which rely on text to label objects distinguish between upper and lower case characters, some do not. To avoid confusion, we strongly recommend using **LOWER_CASE=DISABLED**. When **LOWER_CASE** is disabled, all characters will be converted to upper case before the text component is added to the drawing.

Example: **ADD TEXT=%**

This example uses the macro symbol % to be a placeholder for the text string. You will be prompted to enter the string at the keyboard.

Do not create multiline text components for conversion to maskable polygons. See the lesson on Maskable Text in the Classroom Tutorials Manual.

When typing at the prompt, quotes are not required. The **MULTILINE_TEXT** parameter of the **TEXT** command controls whether or not you can enter more than one line of text. If **MULTILINE_TEXT** is enabled, after typing your text at the prompt, pressing <ENTER> the first time will enter your first line of text, then you will be prompted for the next line. Pressing <ENTER> twice signals the end of input. If **MULTILINE_TEXT** is disabled, the command will complete with the first <ENTER> and you will not be prompted for another line of text. **MULTILINE_TEXT** has no effect when you provide the quoted string(s) in the command.

You can see the default width value associated with each layer by using the **TEMPLATE** command.

The **SIZE** parameter controls the height of the characters created. It must be a positive real number. If the *size* parameter is omitted, the default width assigned to the text layer will be used. This value is set by the **LAYER** command. The actual height of the created characters will be 75% of the *size* parameter or the default width. This prevents the situation shown in Figure 35. Text created with a size equal to the width of a component will label it legibly.

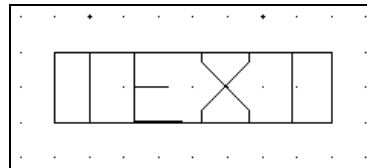


Figure 35: What text would look like if the height of the letters was the same as the width of a wire.

The **LAYER** command associates a default width with a layer.

Example:

ADD TEXT=% LAYER=M1 SIZE=2

ICED™ will prompt you to type the text for this ADD command at the keyboard. The text will be created on layer M1 with a size of 2 user units. If you add this text on top of a wire with a width of 2 user units, it will look like the text shown in Figure 36.

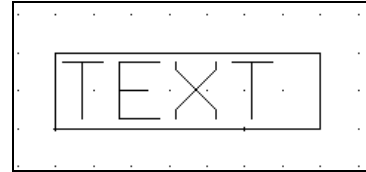


Figure 36: Text of SIZE=2 added on top of a WIDTH=2 wire.

See the **FILL** command to learn about turning fill on and off.

When fill is turned on, the wire shown in Figure 36 might completely hide the text on the same layer if the text were drawn in the same color as the wire. However, ICED™ will draw text in white when fill is turned on, no matter what layer it is on. This way, the text will still be visible when fill is on. When fill is turned off, the text is drawn in the same color as the layer it is on.

You can mirror the text before it is added to the drawing by using either the **MX** or **MY** keywords. **MX** will mirror the text component in the x-direction. The **MY** keyword mirrors the text in the y-direction. Refer to Figure 37 for examples.

See the **ROTATE** command description on page 292 for more details on rotating components.

You have three choices on how your text can be rotated. **HORIZONTAL** (or **R=0**) indicates that the text will be added without rotation. This is the default if you do not provide one of the three choices. **VERTICAL** (or **R=1**) will orient the text so it is read from down to up. When **R=rot_code** is used, *rot_code* indicates the number of 90° counter-clockwise rotations to perform. Legal values for *rot_code* are listed in the ROTATE command description.

The order of the mirror and rotation keywords makes a difference. The command parameters are processed left to right.

How mirrored or rotated text is displayed depends on the current value of the ORIENTATIONS parameter defined by the TEXT command.

The **TEXT** command defines the default value for the JUST parameter.

The **JUST** parameter determines text justification. The value of *just_code* will determine the origin of the component. The location of the origin is stored as the true location of the text. The importance of *just_code* is made clearer in the following examples.

Example:

ADD TEXT="TEXT" JUST=BL

When placing the text for a command using JUST=BL, the origin will be located on the **bottom left** corner of the containing rectangle. The cursor will mark the origin while you place the text. See Figure 38. The coordinates of the point where this corner is placed will be stored as the location of the component.

<i>just_code</i>	Default text justification
LB or BL	B ottom L eft
LC or CL	C enter L eft
LT or TL	T op L eft
CB or BC	B ottom C enter
CC	C enter
CT or TC	T op C enter
RB or BR	B ottom R ight
RC or CR	C enter R ight
RT or TR	T op R ight

Figure 39: Text justification codes.

	No mirroring	MX	MY
Horizontal R=0	ABCD	DCBA	UDCBA
Vertical R=1	UDCBA	UDCBA	UDCBA
R=2	UDCBA	UDCBA	UDCBA
R=3	UDCBA	UDCBA	UDCBA

Figure 37: All possible combinations of mirroring and rotating text. The mirroring keyword came before the rotation parameter in all of these ADD TEXT commands.

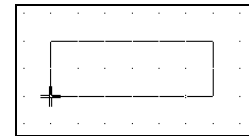


Figure 38: The cursor is placed at the BL corner of the text box.

See Figure 39 for valid values of *just_code*. (The same table is listed in the TEXT command. The TEXT command is used to set the default justification.) These values are compatible with the justification codes in CALMA-GDSII files.

Entering point data is explained in more detail on page 125.

The point on the text component selected by the JUST parameter will be placed at the coordinates indicated by the *pos* parameter. If you do not specify *pos* with the ADD TEXT command, you must place the text component with the mouse.

The coordinate pair defined by *pos* is stored with the component. This point will be used to determine if a text component is attached to other geometry as a label by some types of post-processing software. The location of this point must be **inside** the boundaries of other components to label them.

Use the **TEXT DISPLAY-ORIGINS=ON** command to force text origins to be displayed with small crosses.

The text components in both Figure 40 and Figure 41 store the same coordinates as the location of the components. These coordinates are well inside the outlines of the wires they label. The CC (center) *just_code* parameter forces ICED™ to draw the text components as shown in Figure 40. This can be confusing on a plot. When you use JUST=LC (left center) as shown in Figure 41, the text components are justified appropriately for these wires. LC is the most commonly used justification code.

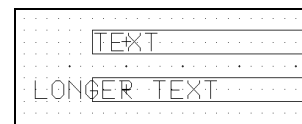


Figure 40: Text components on wires with JUST=CC.

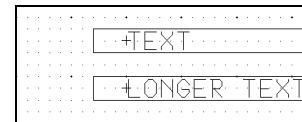


Figure 41: Text components added with JUST=LC.

In any case, you must make sure that the point stored as the location of the text is inside the boundary of the component it labels. The appearance of the text on the screen or in a plot does not insure that the point used to locate the text is really on top of the component it labels.

Values of *just_code* that place the true location of the text on the bottom or top of the text component are especially dangerous. If the true location of the text is right on the edge of a component, rather than inside the boundary (see Figure 42), it is impossible to predict if post-processors will use the text to label to component.



Figure 42: Unwise use of JUST=BL to label a wire

Example: **ADD TEXT "This is some 'sample' text" LAYER=TEXT JUST=CC**

Since the JUST=CC parameter is used, the center of the text component is considered to be the origin when placing the component to your drawing. Since no position is supplied with the command, the mouse will be placed on the center of the bounding box of the component as shown in Figure 43, and you can move the mouse to place the component. Press the left button to add it to the drawing.

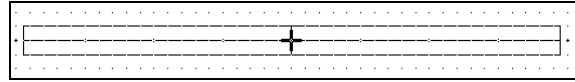


Figure 43: Only the outline is shown while using the mouse to place text.

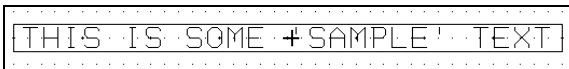


Figure 44: The added text component.

The **LAYER** command associates a default width with a layer.

This text component will be placed on the TEXT layer. Since no SIZE parameter is supplied, the default width parameter for the TEXT layer will be used for the height of the characters. The text will not be rotated since the VERTICAL and R keywords are not used.

ADD CELL=cell_name [MX | MY] [R=rot_code] [[AT] pos]

See page 18 for details on cell files and directories.

ADD CELL adds a cell to the drawing. The cell file must exist in the same working directory as the root cell or in a cell library defined in the current project¹⁹. You cannot include the path to the cell file or the cell file extension.

See the **GROUP** command for details on using **UNGROUP** to explode cells.

Once added to your drawing, a cell can be manipulated just like any other component. You can use the MOVE or ROTATE commands to change its location. The COPY command can be used to place more copies of the same cell, instead of using more ADD CELL commands. The UNGROUP command can be used to replace a cell with the components it contains.

The optional MX and MY keywords in the ADD CELL command will mirror the cell before it is added to the drawing. MX will mirror the cell about the line x=0 in the coordinate system of the cell. The MY keyword mirrors a cell about the

¹⁹ The cell libraries are defined by the environment variable ICED_PATH in your project batch file.

line $y=0$. In other words, MX mirrors the cell about the y-axis, and MY mirrors it about the x-axis.

The optional R keyword allows you to rotate the cell before it is added. *rot_code* indicates the number of 90° counter-clockwise rotations to perform. (All valid values for *rot_code* are listed in the ROTATE command description.)

See the **OUTLINE** command to learn about cell outlines; dotted lines drawn on the bounding boxes of cells.

See Figure 45 for examples of mirroring and rotating cells. The order of the mirror and rotation keywords **does** make a difference. The mirror keyword (MX or MY) was used before the rotation keyword in the examples in the figure.

Text components in the cell may or may not appear as rotated or mirrored depending on the current value of the ORIENTATIONS parameter defined by the TEXT command.

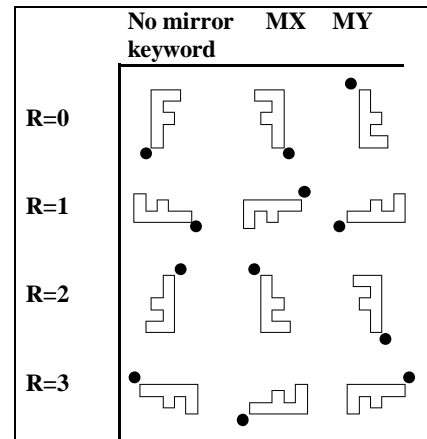


Figure 45: Effects of combinations of rotation and mirror keywords on ADD CELL. The dots represent the origin of each cell.

Entering point data is explained in more detail on page 125.

The AT keyword is not required to enter the *pos* parameter. This coordinate pair defines the point where the origin of the cell will be placed. The origin of the cell may be outside of its bounding box (a bounding box is the rectangle that just covers all components in a cell). *pos* must be a coordinate pair of real numbers. If it is omitted, you must place the cell with the mouse.

Example:

ADD CELL=nand MY R=2 AT (10.5, 102)

In this example, ICED™ will add the cell defined in the file NAND.CEL (or the version of NAND in memory if NAND has already been loaded) to the drawing. Before the cell is added, it will be mirrored about the x-axis and then rotated 180°. The origin of the newly added cell will be placed at the coordinates (10.5, 102.0) of the current cell.

One cell added to another is said to be nested inside it. Cells can be nested up to 16 levels deep.

To edit a cell added to your drawing without exiting, see the **EDIT**, **P_EDIT** and **T_EDIT** commands.

Each cell file contains two databases. The geometric database contains a description of each component; its type, position, select status, etc. The environment database contains information about the cell as a whole; layer names, layer colors, grid settings, etc. When you add a cell to your drawing with the ADD CELL command, ICED™ loads only the cell geometry database. Thus, if layer 1 is defined as red in the cell you are editing but defined as blue in the cell you are adding, the nested cell's environment is ignored and components on layer 1 contained in the nested cell will appear red.

**ADD ARRAY=cell_name N=nx,ny [STEP=disp_x , disp_y] [MX | MY] ...
... [R=rot_code] [[AT] pos]**

How the array is displayed depends on the settings defined by the **ARRAY** command.

The ADD ARRAY command will add an array of cells with the name *cell_name*. The rules that control where ICED™ will search for the cell are the same as those listed above for ADD CELL. The entire array is a single component, It can be moved or deleted as a unit, just like any other component.

(It is not a good idea to create an array with a cell that contains an array. Some post-processing software will not handle this type of array properly.)

The number of cells in the x-direction is given by *nx*. *ny* defines the number of cells in the y-direction. Both parameters must be integers in the range 1:32767. Neither parameter should be omitted. Use a 1 for either value if the array is one-dimensional.

The optional STEP keyword (if used) should directly follow the *ny* parameter. This keyword is used to define the spacing between the cells. If the STEP keyword is used, both *disp_x* (displacement in the x-direction) and *disp_y* (displacement in the y-direction) must be supplied, even if the array is one-dimensional. Either value can be less than the dimension of the cell, so overlapping the cells is possible. If the STEP keyword is omitted, the array will be constructed such that the bounding boxes of the cells will just touch each other.

Example: **ADD ARRAY=letterf N=3,2 STEP=10,15**

This example will create the array shown in Figure 46. The array will be 3 cells wide in the x-direction, and 2 cells high. The origins of the cells will be 10 user units apart in the x-direction and 15 user units apart in the y-direction.

If the *pos* parameter was used in this example, the origin of the array would have been placed at *pos*. Since the position is not supplied, you will place the array with the mouse. The cursor will be at the origin of the array.

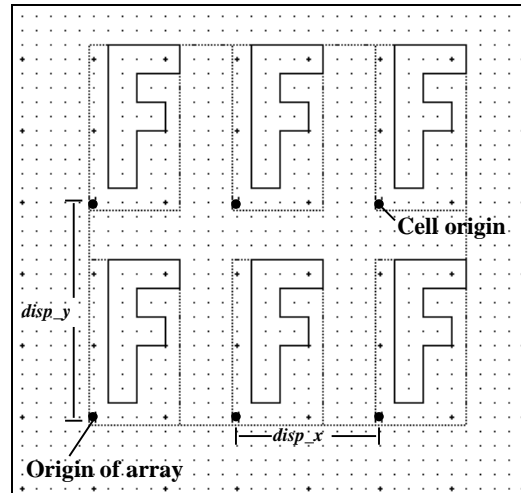


Figure 46

The optional MX and MY keywords can be used to mirror the entire array before it is added to the drawing. MX will mirror the array about the line $x=0$ (in the array's coordinate system). The MY keyword mirrors the array about the line $y=0$.

Example: **ADD ARRAY=letterf N=3,2 STEP=10,15 MX**

This example will add the array shown in Figure 47. Note that the origin of the array is now at the lower right. This is the origin of the first cell in the array.

The optional R keyword allows you to rotate the entire array before it is added to the drawing. *rot_code* indicates the number of 90° counter-clockwise rotations to perform. Valid values for *rot_code* are listed in the ROTATE command description.

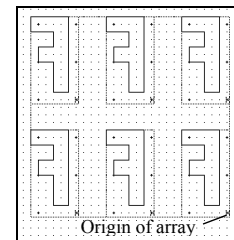


Figure 47: Array created with MX keyword.

If the example above used $R=1$ instead of MX, the array shown in Figure 48 would be created. One counter-clockwise rotation of 90° would be performed.

Defining positions is explained in more detail on page 125.

The *pos* parameter defines the point in the current cell where the origin of the array will be placed. The origin of the array is the origin of the first cell, the cell at the lower left corner of the array before rotation or mirroring.

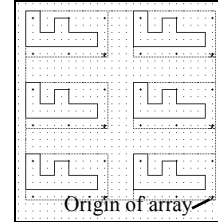


Figure 48: Array created with $R=1$.

Interactions with the SELECT Command.

A SELECT NEW command, executed as the next command after an ADD command, will select the newly added component. Any other components already selected will remain selected.

ARRAY

Set the array display mode.

```
ARRAY [ CELL_LIMIT=cell_limit ] ...  
... [ DRAW_MODE=(BLANK | SIDES | FULL) ]
```

See the **ADD** command to create an array component.

The ARRAY command is used to control the way ICED™ draws arrays. This command affects only the screen display and does not alter the geometry of the design.

Drawing all of the elements in a large array can drastically increase the time ICED™ spends redrawing the screen. You can set the CELL_LIMIT and DRAW_MODE parameters to limit this effect.

See the **BLANK**, **DISPLAY**, **VIEW LIMIT**, and **OUTLINE** commands for other ways to reduce the complexity of the data on the screen.

CELL_LIMIT limits the maximum number of array elements that ICED™ will draw in a single array. If the number of cells in an array exceeds the *cell_limit*, ICED™ will only draw cells on the edges of the array.

The DRAW_MODE places additional restrictions on which array elements will be drawn. The DRAW_MODE settings have the following effects:

- FULL** All the cells that make up an array may be drawn.
- SIDES** Only the cells on the outside edges of an array may be drawn.
- BLANK** The contents of arrays will not be drawn.

Example: **ARRAY DRAW_MODE=FULL CELL_LIMIT=1024**

In this use of the **ARRAY** command, all the cells in a 256 x 1 array will be drawn, all the cells in a 16 x 16 array will be drawn, but only 1024 cells in a 4096 x 1 array will be drawn. If a 100 x 100 array is present, only the edge cells will be drawn. Only 1024 of the cells on the edge of a 1024 x 1024 array will be drawn.

Both the CELL_LIMIT and DRAW_MODE keywords are optional. If either is omitted, the parameter value remains unchanged.

Example: **ARRAY**

This form of the ARRAY command (with no parameters) can be used to display the current values of DRAW_MODE and CELL_LIMIT on the screen.

ARROW

Change function of arrow keys.

ARROW MODE = (PAN | EDIT)

If the **AUTOPAN** mode is ON, moving the cursor off of the screen while digitizing points also pans the view window.

This command lets you choose the function of the arrow and related keys on your keyboard. These keys can be used either for panning the display or for scrolling through the command history. The default is that these keys are used to access the command history. This allows you to repeat commands executed previously with a few keystrokes.

The use of the arrow keys to access the command history is a relatively recent addition to ICED™. Users used to the older versions may feel more comfortable by changing the keyboard to the pan mode with this command. However, even when the arrow keys are in the edit mode, the old panning functions are available by combining the <Ctrl> key with the arrow keys.

The behavior set with this command affects two sets of keys on the average keyboard. The numeric keypad (shown in Figure 49) is one set. These keys will perform the functions listed on the next page as long as the NumLock feature is turned **off**. Most keyboards also have an additional set of arrow keys and <Home>, <End>, <Page Up>, and <Page Down> keys. You can use whichever set is most convenient for you.

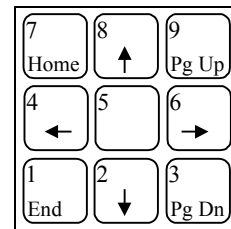


Figure 49:
Portion of
numeric keypad.

Example: **ARROW MODE=EDIT**

This command forces ICED™ into its default keyboard mode. (This command is useful only when you have changed the default keyboard mode with a previous ARROW command.) In the edit mode, the arrow keys scroll through the command history in a similar manner to DOSKEY or CED. The arrow keys can still be used to pan the view window by holding down the <Ctrl> key while pressing the arrow keys.

As long as you not digitizing points with the mouse after entering a command, the behavior of the keys in edit mode is:

<↑> or <Page Up>	Scroll command history up 1 line
<↓> or <Page Down>	Scroll command history down 1 line
<←>	Move cursor 1 space left
<→>	Move cursor 1 space right
<Home>	Move cursor to start of line.
<End>	Move cursor to end of line.

Once you have entered a command that requires you to digitize points, the arrow keys change to pan mode until the command is complete. That is, you do not have to hold down the <Ctrl> key during the execution of a command to pan the view window with the arrow keys even when the keys are in edit mode.

Example: **ARROW MODE=PAN**

This command tells ICED™ to interpret the arrow keys in nearly the same way it did in older versions. In this mode the keys have the following functions before you begin typing a command:

<↑>	Pan up one half of view window
<↓>	Pan down one half of view window
<←>	Pan left one half of view window
<→>	Pan right one half of view window

Once you begin typing on the command line, the arrow keys change to edit mode until you press <Enter>. In other words, while you are typing a command, you can use the arrow keys to scroll through the command history or move the cursor on the command line.

When the arrow keys are in pan mode, you can still scroll through the command history with the <Page Up> and <Page Down> keys. Also, the arrow keys can be used to scroll through the command history by holding down the <Ctrl> key while pressing the arrow keys.

If you can get in the habit of pressing the <Ctrl> key to pan the display, the default arrow key mode of edit is the most useful.

AUTOPAN

Enable or disable automatic view panning.

AUTOPAN [ON | OFF] [PIXELS=*n_pixels*] [SECONDS=*n_seconds*]

The AUTOPAN command is used to control ICED™'s autopan feature. If AUTOPAN is active and you move the cursor beyond the edge of the screen, the view window will move to follow the cursor. The view window will move vertically by steps equal to one half the screen height or horizontally by steps equal to one half the screen width.

Also see the **ARROW** command description to learn how to use the arrow keys to manually pan the view window.

The AUTOPAN feature is active when AUTOPAN is ON and you are digitizing points for any command other than a VIEW command. An autopan is triggered when it is turned on and:

the cursor is moved more than *n_pixels* beyond the layout area of the window

or

the cursor is moved to the edge of the screen and remains there for *n_seconds*.

The value for *n_pixels* must be in the range 0:100. The value for *n_seconds* must be in the range 0.0:10.0. The defaults for the autopan settings are PIXELS=20 and SECONDS=0.2.

The purpose of the PIXELS and SECONDS keywords is to allow you to customize how the autopan is triggered based on your style and speed with the mouse so that unintentional autopans are not triggered. If you find that unintentional autopans are triggered too often, try larger values for *n_pixels* and/or *n_seconds*.

Example: **AUTOPAN**

Using the AUTOPAN command with no parameters toggles between ON and OFF. Thus, if AUTOPAN was enabled before the command above was executed, it will be disabled afterwards.

BLANK and UNBLANK

Hide or show layers or components.

[UN]BLANK ALL
 [UN]BLANK [CELL | ROOT] LAYERS=*layer_list*
BLANK
BLANK SELECT

The [UN]BLANK command is used to blank (make invisible) or unblank (make visible) components or entire layers in a drawing. The BLANK command should not be confused with the DELETE command which actually removes components from the drawing. BLANK affects only the screen display and which components can be selected for modification. It does not affect the mask geometry, plots, or exports to Stream or CIF files.

The [UN]BLANK command combines two conceptually different operations. It blanks or unblanks individual components in the root drawing (i.e. the current cell) **and** controls which cell layers are displayed. These two concepts will be made clear in the following examples.

When you blank a component in the root drawing, a flag is set indicating that the particular component is blanked. There is one such flag for each component in the root drawing. New components are always added to the drawing with their flag cleared. Thus, they will not be blanked unless you execute a blank command after they are added. A blanked component is unselected and can not be re-selected until it is unblanked. Using the UNBLANK ALL, UNBLANK ROOT LAYERS=*layer_list*, or UNBLANK LAYERS=*layer_list* command clears the flags of the affected components.

When you blank a cell layer, a flag is set indicating that all geometry on the cell layer is hidden. There is one such flag for each layer in the drawing. If you add a new cell to the drawing and it has components on blanked layers, they will **not** be displayed. Using the UNBLANK ALL, UNBLANK CELL LAYERS=*layer_list*, or UNBLANK LAYERS=*layer_list* command clears the flags of the affected layers.

The flags set by the BLANK command are saved with the cell file. The components and layers will remain blanked the next time you edit the cell.

[UN]BLANK ALL

Refer to the **PROTECT** command.

The BLANK ALL command blanks all unprotected components in the root drawing **and** all cell layers. UNBLANK ALL unblanks all unprotected components and all cell layers.

Example: **BLANK ALL**

This command will make all unprotected components at the root, and all nested cells, invisible and unavailable for selection. If you add new components they will be visible.

Example: **UNBLANK ALL**

This command will make visible all components at the root and all nested cells.

[UN]BLANK ROOT LAYERS=*layer_list*

Several examples of layer lists are provided later on page 161. See a complete description of layer lists on page 106.

This command blanks (or unblanks) all unprotected components in the root drawing that are on layers specified by *layer_list*. Components on other layers and components on a layer in the layer list, but contained in nested cells, are not affected. The *layer_list* can contain any layer names or layer numbers including layer 0. Unblanking root layer 0 unblanks any blanked cells that have been blanked with BLANK SELECT. However, blanking root layer 0 has no effect.

Examples: **UNBLANK ROOT LAYER 0**
 UNBLANK LAYER 0

Both commands unblank any cells that were previously blanked. If specific cell layers were blanked, they remain blanked after this use of UNBLANK.

[UN]BLANK CELL LAYERS=*layer_list*

To hide the contents of nested cells also see the **DISPLAY** command.

This command blanks (or unblanks) the cell layers specified by the *layer_list*. Components at the root level (not nested in cells) are not affected. The *layer_list* can contain any layer names or layer numbers including layer 0. However, [un]blanking cell layer 0 has no effect. To turn off the display of cell outlines (shown on layer 0) you must use either the OUTLINE or DISPLAY OUTLINE_DEPTH commands.

Example: **BLANK CELL LAYER –M1**

The command above will turn off the display of components on all layers in nested cells **except** for layer M1. In other words, components on layer M1 in nested cells will still be displayed, but all other nested components will not be shown. The display of components in the root cell is not affected.

[UN]BLANK LAYERS=*layer_list*

Not including either the CELL or ROOT keyword in the [UN]BLANK LAYERS command combines the effects of the [UN]BLANK ROOT LAYERS command and the [UN]BLANK CELL LAYERS command.

BLANK

The BLANK keyword, when used without any other keywords, blanks all **new** components in the current cell on layers that are currently blanked. ICED™ keeps a list of root layers that are currently blanked. Components added after a BLANK ROOT LAYERS=*layer_list* command are visible until a BLANK command is issued setting the blank flag of all those new components. (Remember that a new component is always added to the drawing with its blank flag cleared.)

Example: **BLANK ROOT LAYER=M1**
 ADD BOX LAYER=M1 AT (0,0) (10,10)
 BLANK

In this example, the root layer M1 is blanked by the first command. This means that any components in the cell being edited which are on layer M1 will become invisible. (Components nested in other cells on layer M1 are not affected.) The ADD command creates a new component on layer M1. This new component is displayed on the screen. The third command (i.e. BLANK with no other keywords) will make the new component on M1 invisible.

Using the BLANK command without other keywords is very useful after using UNGROUP to ungroup a cell. UNGROUP deletes a cell and adds its components to the cell currently being edited. Since these are new components in the current cell, the components on currently blanked layers will be visible until a BLANK command is used to update their blank status.

BLANK SELECT

This command blanks all fully selected components. Partially selected components are not affected. This command will **not** automatically generate an embedded SELECT command. The components must be selected prior to issuing the BLANK SELECT command.

Example: **SELECT CELL JK_FF ALL**
 BLANK SELECT

The first command selects all instances of cell "JK_FF" added to the cell currently being edited. The second command blanks them. UNBLANK ALL or UNBLANK ROOT LAYER=0 will show them again.

Using Layer Lists in the [UN]BLANK Command

The following four examples illustrate the power of the layer list construct when used with the BLANK command.

Example: **BLANK LAYER PWEL**

This command blanks (makes invisible) all unprotected components in the root drawing that are on layer PWEL. It also prevents ICED™ from displaying any subcell components on layer PWEL. Components on other layers are not affected. Components added to the current cell on layer PWEL after this command has been executed will not be blanked.

Example: **BLANK LAYER -PWEL**

This command blanks (makes invisible) all unprotected components in the current cell **except** those that are on layer PWEL. It also prevents ICED™ from displaying any subcell components that are on any layer other than PWEL. Components on layer PWEL are not affected.

Example: **BLANK CELL LAYERS 1:255**

See **OUTLINE**
for details on
array and cell
outlines.

This command prevents ICED™ from displaying any subcell components on layers 1 through 255. It makes all components in nested cells invisible, with the possible exception of cell and array outlines.

Example: **UNBLANK LAYER M1 + VIA**

This command unblanks any components in the current cell that were on layers M1 or VIA. It also instructs ICED™ to draw components on those layers that are contained in nested cells.

Interactions Between the UNBLANK and SELECT Commands

If you issue a SELECT NEW command just after an UNBLANK command, any newly unblanked components will be selected.

BLINK***Make the indicated color blink on the screen.***

BLINK [OFF] [COLOR=*color_id*] [RATE=*rate*] [TIME=*seconds*]

See the
COLOR
command to
assign names to
colors.

The BLINK command will cause the indicated color to blink at the rate specified for the time period specified.

The *color_id* may be a color number in the range 1:15 or a previously assigned color name. When setting the *color_id* parameter, the COLOR keyword is **not** optional.

The *rate* parameter must be an integer in the range 0:20. The units of this parameter are blinks per second. Using RATE=0 will result in a rate of 1 blink per second, the slowest blink rate.

Seconds is used to determine how long the color will blink. It must be an integer in the range 0:10. Using TIME=0 will cause the color to continue blinking until a BLINK OFF command is executed.

The order of the keywords is unimportant for the BLINK command. If any of the keywords is omitted, the parameters will default to the following values:

COLOR	defaults to 15 (the data highlight color)
RATE	defaults to 10 blinks per second
TIME	defaults to 2 seconds.

Example: **BLINK COLOR=HI T=0**

This command will cause all components on layers drawn with the HI color to blink. The components will continue to blink until a BLINK OFF command is executed.

Note on Color Priority

The BLINK command is intended to be used primarily by the OUTLINER utility that puts temporary geometry in a cell to highlight the path of a net recognized by the NLE utility.

See the
COLOR
command for an
explanation of
color priority.

If you use this command for other purposes, keep in mind that if you blink a low priority color, blinking geometry may be hidden by components drawn with higher priority colors. The BLINK command does not change the priority of the blinking color. If you are using this command in a command file, you may want to copy components to a temporary layer drawn with the HI color (the highest priority color) before issuing the BLINK command.

The OUTLINER utility (described in the NLE and LVS Reference Manual) always draws its components with the HI color. Since this is the highest priority color, the blinking components are never hidden by other geometry.

CIF

Output design as a CIF format file.

CIF SCALE=*cif_scale* [TEXT=*text_code*]

See the **LAYER** command to assign CIF layer names and the **TEMPLATE** command to display the current CIF layer parameters.

See **A Special Note on CIF and STREAM** at the end of the **LAYER** command description.

To import CIF files, use the **UnCIF** utility.

The CIF command is used to output ICED™ design data as a CIF (Caltech Intermediate Form)²⁰ file. All cells nested in the current cell will also be output to the CIF file. Their names will be recorded using the CIF user-9 command.

The CIF standard does not support text, arrays, or flush end (type 0) wires. Several conventions for representing text in CIF are in use. None are universally accepted. (See [TEXT=*text_code*] below.) Arrays and flush end wires are converted to CIF compatible forms with equivalent geometries on export to CIF.

The CIF command will output only components on layers that have been assigned non-blank CIF layer names. **ICED™ deliberately forgets the current CIF layer names each time you open the cell.** If you do not reassign the names before executing the CIF command, you will get the error message: "No layers assigned CIF layer names." This feature is intended to prevent accidents in which CIF files (and mask sets) are created using an obsolete ICED-CIF layer correspondence. A convenient way to assign CIF layer names is to include them in your startup command file. You can execute the startup command file, and thus reassign the CIF layer names, by using the @* command or selecting @START from the menus.

SCALE=*cif_scale*

The CIF specification states that all distances are given in units of centi-microns. The SCALE parameter establishes the number of centi-microns in 1 ICED™ user unit. The *cif_scale* must be an integer in the range 1 : 100,000.

Example:

CIF SCALE=100

²⁰ The CIF standard is defined in "An Introduction to VLSI Systems" by Lynn Conway and Carver Mead, Addison-Wesley Publishing Company, 1980.

This command causes ICED™ to create a CIF file. If the cell you are editing is named NAND, the CIF output file will be named NAND.CIF. The data in the CIF file will be scaled so that:

1 ICED™ user unit = 100 centi-microns = 1 micron.

[**TEXT=***text_code*]

If you do not specify the TEXT parameter in the CIF command, ICED™ text components will not be included in the CIF output file. The only valid values for *text_code* are 2 and 94. Specifying one of these values causes ICED™ to output text using the user-2 or user-94 CIF command.

Example: **CIF SCALE=2540 TEXT=2**

In this example the data will be scaled so that:

1 ICED™ user unit = 2540 centi-microns = 0.001 inch = 1 mil

in the CIF file. ICED™ text components will be included in the CIF file using the CIF user-2 command.

Location of Output File

In previous versions of ICED™, when you executed the CIF command during a nested edit (begun with the EDIT, P_EDIT, or T_EDIT commands), the output file was always written to the directory in which the subcell was stored. This was true even if this directory was a read-only or copy edit library.

This version of ICED™ will create the output file in the subcell's directory only when that directory is defined as a direct edit library. If the subcell's directory is defined as a read-only or copy edit library, the output file is created in the working directory.

The file name will be the cell name with a .CIF extension.

COLOR

Set colors and priority levels.

```
COLOR color_id [ NAME=color_name / NO_NAME ] ...
... [ PALETTE=red_value, green_value, blue_value ] ...
... [ ONE_DOT=(dot_color | UNDEFINED) ]...
... [ FOUR_DOTS=(dot_color1, dot_color2, dot_color3, dot_color4 | UNDEFINED)]...
... [ LEVEL=level ]
```

See the
LAYER
command to
assign colors to
layers.

The COLOR command is used to define colors used in the screen display and on plots. ICED™ supports 16 colors numbered from 0 to 15. Color 0 (BLACK) is the screen background color. Colors 1:15 can be assigned to layers using the LAYER command. (Certain restrictions are placed on the color definitions to ensure compatibility with the plotting software and older versions of ICED™. These restrictions are summarized on page 167.)

Keyword Order

The COLOR command always starts with the keyword COLOR followed by the *color_id*. The remaining keywords are all optional and their order is unimportant. If a keyword is not supplied, its value remains unchanged.

color_id

The *color_id* specifies the color whose parameters you wish to change. It may be a color number in the range 0:15 or a previously assigned color name.

[NAME=*color_name* / NO_NAME]

Interactions
between the
COLOR,
LAYER, and
PLOT
commands are
discussed below
on page 172.

A color name consists of 1:15 characters. Legal characters are A:Z, #, \$, _, and 0:9. The first character cannot be a digit. You cannot assign the same name to two different color numbers or change the names of colors 0:7. (Their names have special meaning to the 8-color plot software.)

The NO_NAME keyword can be used to remove the color name from a color in the range 8:15.

[PALETTE=*red_value, green_value, blue_value*]

The **BLINK**
command
allows you to
make a color
flash
temporarily.

The palette is defined with a set of 3 color values expressed as integers in the range 0:63. You can choose the PALETTE values without regard to the color names. Thus, it is possible, although not normally recommended, to arrange things so that the layers assigned the color named YELLOW actually appear red on the screen. ICED™ uses color 0 (BLACK) as a background color and color 1 (WHITE) as the foreground color. While you cannot rename these colors, you can change their palette values. Thus, you can use any color as a background color.

Examples:

COLOR BLACK PALETTE=(0,0,10)
COLOR BLACK P 10 10 30

The first example will make the screen background dark blue. The second will make a paler blue background. Note that the keyword PALETTE can be abbreviated to P and still be unambiguous because no other keyword for the COLOR command begins with that letter. Note also that the parentheses, commas, and equal sign can be used to make the command more readable, but are not required.

[ONE_DOT=(*dot_color* | UNDEFINED)]

See the COLORS.CMD command file on page 173 to initialize 16 colors with appropriate ONE_DOT parameters.

To plot with more than 8 colors, see the FOUR_DOT parameter on the next page.

This parameter controls how specific color numbers in the range 8:15 are plotted by the PLOT command when the DOTS=1 parameter (the default) is used. (See the description of FOUR_DOTS below when using the DOTS=4 parameter of the PLOT command.) The ONE_DOT parameter does not affect how a color appears on the screen.

Color raster printers (e.g. a color laser jet or ink jet printer) generate plots as a series of dots in one of 8 colors (the 8 colors in Figure 50). No other colors are supported. Older versions of ICED™ enforced this 8-color rule by allowing you to plot only layers that had colors 0:7 assigned to them. Now you can plot a layer that is assigned a color in the range 8:15 by using this keyword to relate one of the 8 supported plot colors to the color number.

The *dot_color* value must be one of the strings shown in Figure 50.

Example:

COLOR 8 NAME=GRAY PAL=42,42,42 ONE_DOT=BLACK

After this command is executed, components on a layer assigned the color GRAY will appear gray in the screen. When these components are plotted with a PLOT command that uses DOTS=1, they will be drawn in black.

Older startup command files may not define ONE_DOT colors. If a color has never had a valid ONE_DOT color defined for it, its value is set to UNDEFINED. If you try to generate a plot with data on a layer that is assigned an undefined color, the PLOT command will fail.

BLACK
YELLOW
GREEN
RED
CYAN
BLUE
MAGENTA

Figure 50:
Valid *dot_color*
plot colors

[FOUR_DOTS= (*dot_color1*, *dot_color2*, *dot_color3*, *dot_color4* | UNDEFINED)]

See the **COLORS.CMD** command file on page 173 to initialize 16 colors with appropriate **FOUR_DOTS** parameters.

This parameter controls how specific colors are plotted by the PLOT command when the DOTS=4 parameter is used. Unlike the ONE_DOT parameter above, this parameter may be defined for any color number. The FOUR_DOTS parameter does not affect how a color appears on the screen.

When a plot is generated with the DOTS=4 option, each pixel in the plot is drawn by 4 non-overlapping pixels on the page. For a pixel of a given color in the plot data, the 4 dots in the "superpixel" drawn on the page can all be the same color, or they can be a combination that results in a unique color. This method allows plots generated with only 8 colors to appear to be made from many colors.

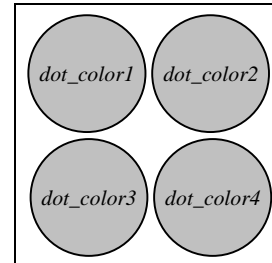


Figure 51: Super-pixel on DOTS=4 plot.

Each *dot_colorn* value must be one of the strings shown in Figure 50 on page 169 or the color WHITE.

Example:

**COLOR 9 NAME=BROWN PAL=32,16,0 ONE_DOT=RED ...
... FOUR_DOTS=(GREEN, RED, RED, YELLOW)**

See the **PLOT** command description for more details on plot options.

The color definition above will result in a brown color on the screen, a red color on plots generated with DOTS=1, and a brown color on plots generated with DOTS=4.

All colors are initialized with UNDEFINED for the value of FOUR_DOTS. The PLOT command will fail if it tries to plot a color with an undefined FOUR_DOTS parameter. However, you can easily assign reasonable values for all colors by using the COLORS.CMD command file described on the next page.

The color WHITE in a FOUR_DOTS definition really means no ink at all. By adding one or more dots of WHITE to a superpixel, you lighten the shade of a color.

Example: **COLOR 8 NAME=GRAY FOUR=BLACK, WHITE, WHITE, BLACK**

This color definition will result in a gray color on plots generated with DOTS=4. Note that a superpixel of this color will be drawn with the two black dots on the diagonal. This is important for consistency when the color is used to draw horizontal and vertical lines. When this superpixel is repeated to form a corner as shown in Figure 53, it forms symmetric lines. If the FOUR_DOTS colors were defined in the order BLACK, BLACK, WHITE, WHITE, the asymmetric lines shown in Figure 54 would be the result.

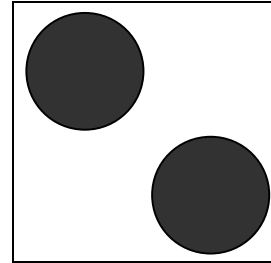


Figure 52:
Superpixel when
FOUR_DOTS =
BLACK, WHITE,
WHITE, BLACK.

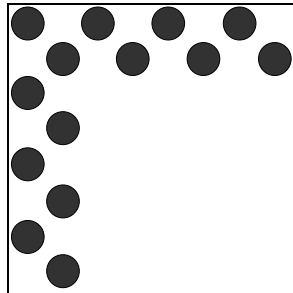


Figure 53

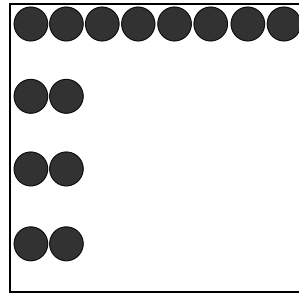


Figure 54

[**LEVEL=level**]

When two colors overlap, the color with the higher level (priority) number covers the color with the lower level number. If the overlapping colors are on the same level, ICED™ attempts to blend them by painting alternating pixels in each color. A similar strategy is used if three or four colors on the same level overlap. If five or more colors on the same level overlap, the overlap area is painted WHITE in the screen display and BLACK on plots.

Color 0 (BLACK, drawn as white on plots) is always a background color, so its level is always 0. Color 1 (WHITE) is used to draw select marks, and text and line components when fill is on. These uses require that WHITE act as a foreground color, thus its level is fixed at 16. All other colors can be assigned levels from 1 to 15.

Color 15 has special purposes in highlighting errors and nodes. While you can reassign its level number, it is best to leave the special level number 17 assigned to it.

Interactions Between the COLOR, LAYER, and PLOT Commands

The palette values assigned with the PALETTE keyword have no effect on the color used in plots. Plots use the colors defined with the ONE_DOT or FOUR_DOTS keywords of the COLOR command to determine the plot color of a specific color number. The color number is assigned to a layer with the COLOR keyword of the LAYER command.

See the table below for defaults on the name, level, and plot colors for the 16 colors in the NEW.CMD startup command file supplied with the installation.

Number	Name	Levels	Uses	ONE_DOT default	FOUR_DOTS default
0	BLACK	0	Screen Background	WHITE	White (x 4)
1	WHITE	16	Data, text components, and select marks	BLACK (text and 5+ color overlaps are also plotted in black)	Black (x 4)
2	YELLOW	1:15	Data	YELLOW (with black outlines)	YELLOW (x 4)
3	GREEN	1:15	Data	GREEN	GREEN (x 4)
4	RED	1:15	Data	RED	RED (x 4)
5	CYAN	1:15	Data	CYAN	CYAN (x 4)
6	BLUE	1:15	Data	BLUE	BLUE (x 4)
7	MAGENTA	1:15	Data	MAGENTA	MAGENTA (x 4)
8-14	user definable	1:15	Data	(See NEW.CMD)	(See NEW.CMD)
15	user definable	1:17	Highlighting data	(See NEW.CMD)	(See NEW.CMD)

Figure 55: ICED™ colors in distribution copy of Q:\ICWIN\TECH\SAMPLES\NEW.CMD

The COLORS.CMD Command File

To learn about startup command files, see page 25.

The file COLORS.CMD is included with the ICED™ installation. This command file will define 16 colors with all appropriate parameters. You may want to add a command that executes this file (or a customized version of this file) to your startup command file so identical colors are defined for each new cell.

! Avoid name conflicts by erasing names of existing colors.

```
COLOR 8 NO_NAME;    COLOR 9 NO_NAME;    COLOR 10 NO_NAME;   COLOR 11 NO_NAME;
COLOR 12 NO_NAME;   COLOR 13 NO_NAME;   COLOR 14 NO_NAME;   COLOR 15 NO_NAME;
```

! Define new color and palette values.

```
COLOR 0      NAME=BLACK  PALETTE=( 0, 0, 0)
COLOR 1      NAME=WHITE  PALETTE=(63,63,63)    LEVEL=16
COLOR 2      NAME=YELLOW PALETTE=(63,63, 0)    LEVEL= 6
COLOR 3      NAME=GREEN  PALETTE=(21,63, 0)    LEVEL= 6
COLOR 4      NAME=RED    PALETTE=(63, 0,21)    LEVEL= 8
COLOR 5      NAME=CYAN   PALETTE=( 0,42,42)    LEVEL=10
COLOR 6      NAME=BLUE   PALETTE=( 0, 0,63)    LEVEL=10
COLOR 7      NAME=MAGENTA PALETTE=(63, 0,63)    LEVEL= 8
COLOR 8      NAME=GRAY   PALETTE=(42,42,42)    LEVEL=14
COLOR 9      NAME=BROWN  PALETTE=(32,16, 0)    LEVEL= 8
COLOR 10     NAME=ORANGE PALETTE=(63,31, 0)    LEVEL= 8
COLOR 11     NAME=PURPLE PALETTE=(21, 0,14)    LEVEL= 3
COLOR 12     NAME=DIM_RED PALETTE=(22, 0, 0)    LEVEL= 3
COLOR 13     NAME=DIM_BLUE PALETTE=( 0, 0,22)    LEVEL= 3
COLOR 14     NAME=DIM_GREEN PALETTE=( 0,22, 0)    LEVEL= 3
COLOR 15     NAME=HI      PALETTE=(63,63,63)    LEVEL=15
```

! Define one_dot and four_dot "palettes" for plotting

```
COLOR BLACK    ONE_DOT=WHITE    FOUR_DOTS=(WHITE, WHITE, WHITE, WHITE)
COLOR WHITE    ONE_DOT=BLACK    FOUR_DOTS=(BLACK, BLACK, BLACK, BLACK)
COLOR RED      ONE_DOT=RED      FOUR_DOTS=(RED, RED, RED, RED)
COLOR YELLOW   ONE_DOT=YELLOW   FOUR_DOTS=(YELLOW, YELLOW, YELLOW, YELLOW)
COLOR CYAN     ONE_DOT=CYAN     FOUR_DOTS=(CYAN, CYAN, CYAN, CYAN)
COLOR BLUE     ONE_DOT=BLUE     FOUR_DOTS=(BLUE, BLUE, BLUE, BLUE)
COLOR GREEN    ONE_DOT=GREEN    FOUR_DOTS=(GREEN, GREEN, GREEN, GREEN)
COLOR MAGENTA  ONE_DOT=MAGENTA  FOUR_DOTS=(MAGENTA, MAGENTA, MAGENTA, MAGENTA)
COLOR GRAY     ONE_DOT=BLACK    FOUR_DOTS=(BLACK, WHITE, WHITE, WHITE)
COLOR BROWN    ONE_DOT=RED      FOUR_DOTS=(GREEN, RED, RED, YELLOW)
```

(continued on next page)

```

COLOR ORANGE      ONE_DOT=RED      FOUR_DOTS=(RED,  YELLOW, YELLOW, RED)
COLOR PURPLE      ONE_DOT=MAGENTA  FOUR_DOTS=(BLUE,  MAGENTA, MAGENTA, BLUE)
COLOR DIM_RED     ONE_DOT=RED      FOUR_DOTS=(RED,   WHITE,  WHITE,  WHITE)
COLOR DIM_BLUE    ONE_DOT=BLUE     FOUR_DOTS=(BLUE,  WHITE,  WHITE,  WHITE)
COLOR DIM_GREEN   ONE_DOT=GREEN    FOUR_DOTS=(GREEN, WHITE,  WHITE,  WHITE)
COLOR HI          ONE_DOT=BLACK    FOUR_DOTS=(BLACK, BLACK, BLACK, BLACK)

! Suggested uses in CMOS processes.
!
! Use high-level (foreground) colors WHITE and GRAY for contact layers.
! Use low-level low-intensity (background) colors PURPLE, DIM_RED, DIM_BLUE,
! and DIM_GREEN for wells and selects.
! Use low- to medium-level high intensity colors for diffusion layers.
! Use medium- to high-level high intensity colors for interconnect layers.
! Reserve color 15 for node outliner.

$$ COLORS.CMD successfully completed

```

Figure 56: COLORS.CMD command file

The two blocks of COLOR commands in the command file above are separated only for readability. You can define all parameters for a color in the same command, or with several separate commands.

You can execute the command file above by executing the following command:

Example: **@COLORS**

See the
@file_name
 command
 description to
 learn more
 about command
 files.

COPY

Copy existing components.

COPY [MOVE] [X | Y] [[BY] *disp*]
COPY MIRROR (X=*x_coord* | Y=*y_coord*)
COPY ROTATE=*rot_code* [[AT] *axis_pos*]

The **SELECT** command can be used to select multiple components prior to using the COPY command.

The COPY MOVE, COPY MIRROR, and COPY ROTATE commands are used to make copies of all fully selected components and move them to a new location. Partially selected components are unaffected. The MOVE keyword is optional. Using COPY without the MIRROR or ROTATE keywords, results in a COPY MOVE command.

If no components are selected prior to executing the COPY command, an embedded SELECT NEAR command is executed allowing you to select a component for the copy.

COPY [MOVE] [X | Y] [[BY] *disp*]

The COPY MOVE command is used to make copies of all fully selected components and move them by a specified displacement from their original location. Partially selected components are unaffected

If you type the COPY command from the keyboard (or in a command file) you can include the numerical value of *disp* as part of the command.

Example:

COPY MOVE BY (2.5, 10.0)
COPY MOVE (2.5, 10.0)
COPY 2.5 10.0

These three examples are equivalent. Each makes copies of all fully selected components and then moves the copies by 2.5 units in the X direction and 10 units in the Y direction.

Also see the **MOVE** command.

If you do not specify the displacement as part of the COPY MOVE command, or if you select the command from the menus, you will have to enter the *disp* parameter with the mouse. If you use the mouse, the displacement is entered by selecting two points. Any two points can be used, but the result will be easier to visualize if you put the first point on a selected component and the second point where you want it to end up.

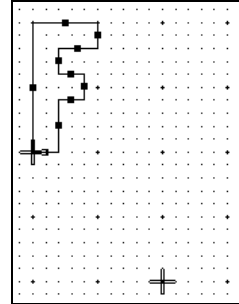


Figure 57: Using the mouse to define coordinates in COPY MOVE.

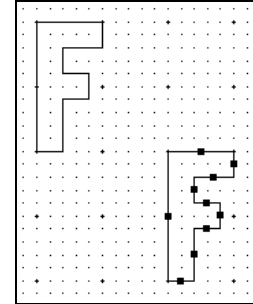


Figure 58: Result of COPY MOVE.

The differences between the X values of the two points and the Y values of the two points become the X and Y displacements.

If you specify a single axis with the X or Y keywords, then displacement along the other axis is ignored. Using the X or Y keywords tells ICED™ to restrict the move to one axis.

Example: **COPY Y 10.0**

This example makes copies of all fully selected components then moves the copies by 10 units in the Y direction.

COPY MIRROR (X=x_coord | Y=y_coord)

The **MIRROR** command moves components.

The **COPY MIRROR** command makes copies of all fully selected components and mirrors the copies into a new position. The axis of the mirror is either a vertical line: MIRROR X, or a horizontal line: MIRROR Y.

Example: **COPY MIRROR Y=10.0**

This command makes copies of all fully selected components and then mirrors them about the line Y=10.0.

If you do not specify the mirror position as part of the COPY MIRROR command or if you select the command from the menus, you will have to enter the mirror position with the mouse. The following command expects mouse input:

Example:

COPY MIRROR X

In this example, the polygon in Figure 59 was selected prior to executing the COPY command. Since the *x_coord* parameter was not provided in the COPY command, the vertical mirror line appeared on the screen to define the copy axis. The mirrored polygon is shown in Figure 60.

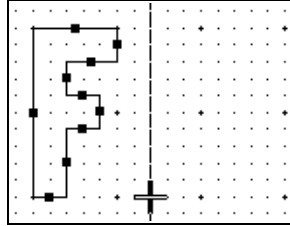


Figure 59: Using COPY MIRROR X command on a polygon.

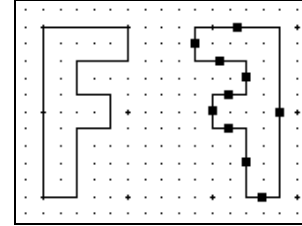


Figure 60: The effect of COPY MIRROR X.

COPY ROTATE=rot_code [[AT] axis_pos]

See the **ROTATE** command to rotate components without copying them.

The COPY ROTATE command makes copies of all fully selected components and rotates the copies around a point. The rotation is restricted to multiples of 90°. *rot_code* indicates the number of counter-clockwise 90° rotations to perform. Legal values for the *rot_code* parameter are listed in Figure 61.

<i>rot_code</i>	counter-clockwise rotation	result
0	0°	F
1	+ 90°	⌊
-1	- 90°	⌋
2	+180°	⌌
-2	-180°	⌍
3	+270°	⌎
-3	-270°	⌏

Figure 61: ICED™ Rotation codes.

Example:

COPY ROTATE 1 AT (10, 20) COPY ROT-3 (10 20)

Both commands make copies of all fully selected components and rotate them by 90° counter-clockwise about the point (10,20).

If you do not specify the axis position as part of the **COPY ROTATE** command or if you enter the command from the menus, you will have to enter the axis position with the mouse. The following command expects mouse input:

Example: **COPY R=2**

Note that the ROTATE keyword can be abbreviated to the single letter R.

Interactions Between the COPY and SELECT Commands

If you issue a COPY command and no components are selected, ICED™ will automatically generate an embedded SELECT NEAR command. After the COPY command is complete, all components will be unselected. If you then issue a SELECT NEW command, ICED™ will select the newly created component(s).

If components were selected prior to executing the COPY command, the original unmoved components will be unselected when the COPY is complete, and the newly created components will be selected. If you then issue a SELECT NEW, ICED™ will select the original components as well.

If you try to issue a COPY command that moves a copied component outside the drawing boundaries, the copy command will fail. You can use the SELECT FAIL command to select the components that caused the failure and unselect all other components. The SELECT POP command can be used to retrieve your original selections.

CURSOR

Set position cursor type.

```
CURSOR [ cursor_type ] [ SNAP=(ON | OFF)]
```

When you use the mouse or other pointing device to enter a coordinate, ICED™ marks its position by drawing a small X on the screen. The X moves as you move the mouse. This X is called the position cursor.

See the **SPACER** command to learn about the special wiring guide cursor available when digitizing the points of a wire or polygon.

ICED™ actually supports five different position cursors. In general, cursor type 1 is most useful with standard VGA displays. However, if you are using a high resolution display or are working in very dense areas, cursor 1 may be too light. In these situations, you may wish to use the CURSOR command to set the cursor type.

Type	Description	Graphic
1	Light X	×
2	Light cross	⦶
3	Heavy X	⊗
4	Heavy cross	⦶
5	Heavy arrow	➤

Figure 62: ICED™ Cursor types

The snap grid defines the points where positions can be digitized with the mouse. It is set with the **SNAP** command.

In the original DOS version of the program the cursor would always snap to the closest point on the snap grid as you moved the mouse. This makes it easier to digitize points since the cursor is actually located where the point will be digitized when you click the left mouse button. However, this behavior was very difficult to implement in the Windows version and early versions did not have this feature.

The newer Windows versions support a SNAP mode. The default mode is now SNAP=ON. At fine scales, this makes the cursor “jump” from snap point to snap point as you move the mouse during most commands. If you prefer to have the cursor move smoothly, use SNAP=OFF.

CUT

Cut a box, polygon, wire, or line into two parts.

CUT (X [=x_coord] | Y [=y_coord])

Use the
MERGE
command to
rejoin cut
components.

The CUT command divides selected boxes, polygons, wires, and lines into two parts. The division takes place along the vertical cut line $X=x_coord$, or the horizontal cut line $Y=y_coord$. After the cut is complete, the resulting parts are unselected.

If no components are selected prior to executing the CUT command, an embedded SELECT SIDE IN is generated to allow you to select the sides to cut. Use the mouse to define a box. All sides either partially or completely inside the box will be selected.

The cut line should intersect **exactly two** selected sides of a box or polygon, or **exactly one** selected segment of a wire or line. If the cut line intersects more than the required number of selected sides or segments, ICED™ may consider the cut ambiguous and report an error.

If you do not supply the coordinate of the cut line in the command, you can define it with the mouse.

Example:

CUT Y

Since no y_coord is provided with this use of the CUT command, a horizontal cut line will appear on the screen. Use the mouse to move the cut line and determine where to cut the selected components.

Refer to the
RESOLUTION
command for
details on the
resolution grid
and the
resolution mode.

In general, the coordinates of the cut objects may or may not be on the resolution grid. Off grid coordinates can arise if you cut a polygon side or wire segment that is not at 0°, 45°, or 90°. The way the CUT command treats off grid coordinates depends on the current setting of the resolution mode.

If the resolution mode is HARD, then ICED™ will round the coordinates to be on the resolution grid. If the resolution mode is SOFT, ICED™ will only round any new vertices to the nearest database unit.

DELETE

Remove all fully selected components from the drawing.

DELETE

The DELETE command deletes all fully selected components from the drawing. It does not affect unselected or partially selected components.

If you issue a DELETE command and no components are selected, ICED™ will automatically generate an embedded SELECT NEAR command allowing you to select the components to delete.

Example:

**SELECT PARTS ALL
DELETE**

The first command fully selects any components which were partially selected. The second command deletes all fully selected components.

DISPLAY

Control how ICED™ displays data.

```
DISPLAY [ CELL_DEPTH=cell_depth ] ...
... [ CELL_LABELS=(ON | OFF) ] ...
... [ OUTLINE_DEPTH=out_depth ] ...
... [ CURSOR=cursor_no ]
```

The DISPLAY command sets a variety of options that control the way ICED™ displays data.

[CELL_DEPTH=*cell_depth*]

See the **ARRAY** and **VIEW LIMIT** commands for other ways to limit the complexity of the display.

All components in the current cell are said to have depth 0. Components contained in depth 0 cells are said to have depth 1. Components in depth '*n*' cells have depth '*n*+1'. The cells that form an array of depth *n* have depth '*n*+1'.

The CELL_DEPTH parameter limits the depth to which ICED™ will display components in nested cells. ICED™ will only draw components whose depth is less than or equal to the *cell_depth*. A cell at depth *cell_depth* will be displayed only as a cell outline. Components nested in the cell will not be displayed.

The DEPTH keyword is a synonym for the CELL_DEPTH keyword.

[CELL_LABELS=(ON | OFF)]

If the CELL_LABELS mode is ON, the name of a cell will be displayed within the cell outline if the depth of the cell is such that the contents of the cell are not shown but the cell outline is displayed. If the scale is too small to display the text, the name will not be shown until you zoom in to increase the scale.

The keywords LABELS and CELL_NAMES are synonyms for the CELL_LABELS keyword.

[OUTLINE_DEPTH=*out_depth*]

The **OUTLINE** command can also be used to set this parameter.

ICED™ can draw dotted white bounding boxes around any unblanked cells and arrays. The OUTLINE_DEPTH parameter is used to set the depth for which ICED™ draws these bounding boxes.

ICED™ will draw bounding boxes for unblanked cells and arrays that have a depth **less than** *out_depth*. *out_depth* must be an integer in the range 0:100.

[CURSOR=*cursor_no*]

The **CURSOR** command can be used to set this parameter. Refer to this command for a table of cursor types.

When you use the mouse or other pointing device to enter a coordinate, ICED™ marks its position by drawing a small X on the screen. The X moves as you move the mouse. This X is called the position cursor. One of 5 differently shaped position cursors can be selected with DISPLAY CURSOR. The *cursor_no* must be an integer in the range 1:5.

Keyword Order

The order of keywords for the DISPLAY command is unimportant. All of the keywords are optional. If any (or all) of them are omitted, the parameter values remain unchanged.

Example:

DISPLAY

Use this form of the command to determine the current DISPLAY parameters.

DOS

Launch console application and wait

DOS [-]

or

DOS [^][-]*dos_cmd_string*

The **SPAWN** command will open an MS-DOS console window that will remain open while you continue to work in the editor.

The DOS command is used either to execute DOS commands (usually from inside a command file) or to open a separate console window running the MS-DOS command interpreter.

This command halts the layout editor until the DOS command *dos_cmd_string* is complete, or until the new window is closed with the EXIT command. If you prefer to keep working in the layout editor while the task begun with this command is executing, use the **SPAWN** command instead.

DOS

When you use the DOS keyword with no command string, a new window is created running the DOS command interpreter.

Any changes to the current directory, disk, or environment variables in this window will not affect settings in the layout editor.

Examples:

DOS

This command will open a new window running the MS-DOS command interpreter. By typing a valid MS-DOS command at the prompt you can execute the ICED™ utilities, DOS commands, and batch files or other programs. Type "EXIT" at the console prompt when you are done to close the window and return control to the layout editor.

Changes to the Environment

Both forms of the DOS command will by default add the ICED™ installation directory to the front of the system PATH environment variable. This means that programs stored in this directory can be executed without typing the path to the executable file. Also, if two different executable files exist with the same name, where one copy is on your default PATH, and the other copy is in your ICED™ installation directory, the file in your ICED™ directory will be executed.

On Windows NT, the PATHEXT environment variable is also modified. This variable lists valid DOS console application file extensions. PATHEXT is temporarily set to ".COM;.EXE;.BAT". This is done to prevent Windows/NT from treating .CMD files as executable files.

If your system crashes while you work in the new window (or for any other reason), you can use the journal file to recover your work. See Appendix B.

The other environment variables created for ICED™ and its utilities (e.g. ICED_PATH, ICED_HOME, etc.) are defined and can be used by any programs executed from the DOS console window or by the *dos_cmd_string*.

All of these changes to your default DOS console environment are temporary and local. If you launch an MS-DOS console window with any non-ICED™ method (e.g. Start->Programs->MS-DOS prompt), then you will not see these changes to the environment.

Type the MS-DOS command SET to see the current environment.

In Windows 95 and Windows 98, these environment changes must be made with a batch file, since these versions of Windows do not support a direct method of controlling the environment of a console window. In these operating systems a batch file with the name W95\$ENV.BAT will be created in the Q:\ICWIN²¹\TMP directory. ICED™ will execute this file to create the appropriate environment prior to opening the console window or executing *dos_cmd_string*.

²¹ Remember that Q:\ICWIN represents the drive and path where you have installed ICED™.

The Optional – Prefix

If you prefer to avoid the execution of the W95\$ENV.BAT file in Windows 95 or Windows 98, you can add the '-' prefix to the command. This suppresses execution of the batch file and the creation of the console window when *dos_cmd_string* is supplied in the command. The environment will not be customized by ICED™.

The '-' prefix is intended only for launching Windows GUI (Graphical User Interface) programs. If *dos_cmd_string* refers to a console-based application (e.g. utilities supplied with MS-DOS that require a character-based command interpreter) then **do not** include the '-' in the command.

DOS [^][-]*dos_cmd_string*

The four valid quote characters are ", ', ~, and `.

You can issue a single DOS command by typing DOS followed by a command. You should use quotes around the *dos_cmd_string* (including any optional prefixes) if the command contains any syntax that may confuse the command parser (e.g. use of '&' or '+', or other quotes).

ICED™ will issue the *dos_cmd_string* command and when it completes control will return to the layout editor. If *dos_cmd_string* is not preceded by a ^, ICED™ will create a visible console window while the command executes. When the DOS command is complete, the window will close.

There will be no pause between completion of the command and the return to ICED™. This means that for some commands (e.g. the DOS DIR command) you will not have time to view the output before ICED™ refreshes the screen. Use the form of the ICED™ DOS command without *dos_cmd_string* for this type of DOS command so you will have a chance to view the result.

It is more common to use this form of the DOS command when you are creating a command file that needs to call another program. For example, if you need to delete a file from within your command file you could use the following command:

Example: **DOS ^DEL %TMP^MYTEMPFILE.TXT > NUL**

See the **SHOW** command (in this manual) or the Command File Programmer's Reference Manual for more details on system macros.

The first '^' symbol in the command minimizes the temporary console window. If it were not present, a console window would flicker briefly on the screen. The second '^' symbol is used to delimit the system macro %TMP (which contains the path of the Q:\ICWIN\TMP subdirectory) from the name of the file you wish to delete.

The "> NUL" at the end of the command takes any output from the command and redirects it to the NUL device. In other words, the output is ignored by the system. This is important for commands where you use the '^' prefix to avoid creation of the console window. If output is generated by the command (e.g. "File not found") and it cannot print to the console window, it can create odd effects like pausing the system until a key is pressed. You can replace the "NUL" with a valid file name if you would prefer to log the result of the command in a file that you can use to debug your command file.

If you used the '-' in the command above, the command would fail since the DEL.EXE program requires the MS-DOS console command interpreter.

Example:

DOS -CALC

This command will launch the CALC.EXE interactive calculator program available with most Windows installations. Since this is a GUI program, it does not require a console window. The '-' in the command prevents the creation of the unnecessary console window.

You can perform any number of calculations, then close the calculator window to return to the layout editor. If you want the calculator window to remain open while you continue to work in the layout editor, use the **SPAWN** command instead.

See the **KEY** command to assign commands to keyboard keys.

You can assign this command to a key so that you can open the calculator utility with a single keystroke.

DRC

Create an intermediate file for the DRC program.

```
DRC [ ARRAY_MODE=( FULL | SIDES | BLANK ) ]
    [ BORDER=border | RULES_FILE=file_name ]
    [ALL | SELECT | IN [= pos1 pos2 ] ]
```

See page 191 for the location of the output file when the current cell is in a protected library.

The DRC command is used to output design data in the format required by the DRC (Design Rules Checker) or NLE (Net List Extractor) programs. The DRC is used to verify your design with technology dependent design rules. The NLE program extracts net and device data from your design to pass to the LVS (Layout Vs. Schematic) program. These are all stand-alone programs sold separately by IC Editors, Inc.

The name of the file created by the DRC command will be *cell_name*.POK, where *cell_name* is the current cell name. The file is usually located in the same directory as the current cell. This file is not an ASCII file and cannot be edited.

[ARRAY_MODE=(FULL | SIDES | BLANK)]

The DRC command does not need to be executed explicitly when you use the new internal DRC features available by loading the special DRC menu. See the **Classroom Tutorials Manual**.

This keyword determines how arrays are output for design checking. If ARRAY_MODE=FULL, then all elements of arrays will be output. This is the default. If SIDES is used, only the components in cells on the outer borders of an array will be included in the .POK file. BLANK will prevent any array elements from being included.

The size of the data passed to the DRC for processing can vary greatly with these three options. BLANK will result in the fastest DRC run time, but array cells will not be verified. For preliminary checking, you may prefer to use SIDES. This will avoid most repetitive checking of each cell in an array when the DRC program uses the .POK file. However, for a complete test of your design you **must** use ARRAY_MODE=FULL.

It is very important to perform a final test on your design using `ARRAY_MODE=FULL`. This is the only way to find certain catastrophic errors. An error as simple as one stray wire over the center of an array can change your design from leading edge technology to scrap metal.

[**BORDER**=*border* | **RULES_FILE**=*file_name*]

The **BORDER** and **RULES_FILE** keywords are compatible only with the **IN** keyword discussed below. If **ALL** or **SELECT** is used instead, these keywords should **not** be used. If the **IN** keyword is used, one of these two keywords **must** precede it.

The **IN** keyword will output for checking all components inside a box. To accurately check the components at the edge of this box, the DRC needs data for all components outside of but near the edge of the rectangle. For an example, Figure 63 represents a portion of a design with a corner of the **IN** box shown as a dotted line. The wire labeled **BUS1** is inside the box so it will be included in the data to be checked. **BUS0** is outside of the box, so it will not be included in the data provided to the DRC. Note that **BUS0** has been shifted rather close to **BUS1**. If **BUS1** is now so close to **BUS0** that it violates a wire spacing rule, the error will not be found unless the **BUS0** wire is included in the DRC check. To insure that all components inside the **IN** box are completely checked, you must include components within a border around the **IN** box.

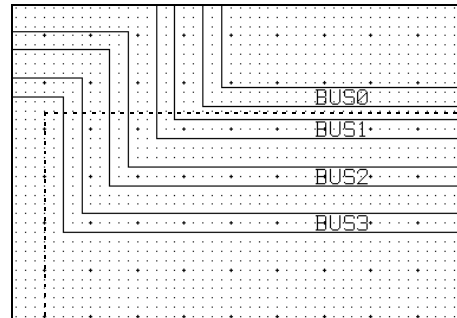


Figure 63: The dotted line represents the edge of the IN box for DRC checking.

In the unlikely event that you wish to use the IN keyword when creating a file for NLE processing, use BORDER=0.

The BORDER parameter allows you to set this border to any distance from the edge of the IN box. *border* must be a positive real number of user units. A *border* of 0 is valid, but errors like the one explained above will not be found by the DRC.

A BORDER parameter is included in the rules file used by the DRC program. If you wish to use the *border* value stored in your rules file, you can use the RULES_FILE=*file_name* parameter instead of the BORDER parameter. If you use RULES_FILE, ICED™ will read this value from the compiled rules file *file_name*.BB.

You can specify the path (or paths) to your rules files in the project batch file so that you do not need to type the path to *file_name* in the DRC command. The rules file search path is defined by the DRC_PATH environment variable.

[ALL | SELECT | IN [= *pos1 pos2*]]

This choice of parameters determines what portion of your data to pass to the DRC. ALL (the default) will include your entire design. SELECT will include only selected components. IN will include all components inside of a box whose corners are given by *pos1* and *pos2*, where *pos1* and *pos2* are coordinate pairs of real numbers.

If components are already selected when DRC SELECT is executed, those components will be output to the .POK data file. If no components are selected, an embedded SELECT IN will be generated by ICED™. All components inside the box you draw on the screen with the mouse will be included in the .POK file. The DRC SELECT command is intended to be used for passing data to the DRC for manipulation rather than for design rules checking. If you wish to generate device layers from design layers, you could select the necessary components, then let the DRC perform the layer generations for you. For design rules checking, use either the ALL or IN keywords instead of SELECT.

If the IN parameter is used, *pos1* and *pos2* can be defined with the mouse if not typed in the command. Remember that either the BORDER or RULES_FILE parameter is required when IN is used.

When either the IN or SELECT keyword is used, it must be the last keyword in the DRC command.

Location of Output File

In previous versions of ICED™, when you executed the DRC command during a nested edit (begun with the EDIT, P_EDIT, or T_EDIT commands), the output file was always written to the directory in which the subcell was stored. This was true even if this directory was a read-only or copy edit library.

The working directory is the directory containing the root cell file specified when you opened the editor.

This version of ICED™ will create the output file in the subcell's directory only when that directory is defined as a direct edit library. If the subcell's directory is defined as a read-only or copy edit library, the output file is created in the working directory.

EDIT

Edit another cell.

EDIT [NOW] [LOCAL_COPY=(TRUE | OK| FALSE)] ...
 [VIEW_ONLY=(TRUE | FALSE)] ...
 ... (CELL=*cell_name* | SELECT | NEAR *position*)

The EDIT command allows you to suspend work on the cell you are currently editing and edit another cell. (We will call these cells the parent cell and the child cell. We will call the cell you were editing when you first entered ICED™ the root cell.) The child cell can be a subcell of the parent cell, an existing cell that is not part of the parent cell, or a new cell. The child cell can **not** be (or contain) the parent cell or another suspended cell (i.e. a cell opened for editing during the current session and not yet closed). You can have up to six suspended cells at one time.

Each cell file contains two databases. The geometric database contains a description of each component; its type, position, select status, etc. The environment database contains information about the cell as a whole; layer names, layer colors, grid settings, etc. When you use the EDIT command to enter a cell, ICED™ loads only the cell geometry database. The initial view window is taken from the child cell's environment database. With this exception, the child cell's environment database is ignored. Thus, if layer 1 is red in the root cell and blue in the child cell it will appear red as you edit the child cell

Unlike the T_EDIT and P_EDIT commands, EDIT will use the child cell's coordinate system. That is, the origin of the cell you are editing will be at the coordinate (0,0) while you are editing it. T_EDIT (transform edit) is much the same as EDIT, except that the parent cell's coordinate system is used instead of the coordinate system of the cell you are editing. P_EDIT (edit in **p**lace) will allow you to edit the child cell while the parent cell is still displayed on the screen and the coordinate system is still that of the parent cell. See the following table for more comparisons of the three edit commands.

The name of the cell you are editing is shown in the upper left corner of the view window. All open cells are listed to the left of the name of the current cell.

Feature	EDIT	T_EDIT	P_EDIT
Coordinate system	Child cell's	Parent cell's	Parent cell's
Parent cell remains on display	No	No	Yes
Disabled commands	none	CIF, STREAM, and DRC	CIF, STREAM, DRC, EDIT, T_EDIT, and P_EDIT
Ability to suspend (edit another cell without exiting)	Yes	Yes	No
Hierarchical editing considerations	Similar to separate ICED™ session, except that the parent's environment database is used (grid, layer parameters, etc.)	The SELECT keyword makes it easy to traverse one level of hierarchy at a time because view of cell will not shift	Using the NEAR keyword will allow you to edit a deeply nested cell while the geometry of the parent remains on screen for reference
Which cells can be edited	Any, including new cells	Only nested cells	Only nested cells

Figure 64: Comparison of ICED™ editing commands.

When you are done editing the child cell, you can use the QUIT, EXIT, or LEAVE commands to return to the parent cell. If you QUIT the child cell, any changes you made will be lost. If you use EXIT or LEAVE, ICED™ will save your changes in RAM. (LEAVE saves the cell only if geometry has been modified.) These changes will be written to disk when you terminate ICED™. This is true even if you eventually QUIT the root cell.

See Journaling and Data Recovery for details.

ICED™ does not update your cell files until you terminate ICED™. Thus, if your system crashes, <CTRL><C> is pressed, or you execute the JOURNAL

command, the cell files will not have been saved. However, your work is not lost. ICED™'s automatic journaling feature will allow you to recover it.

See page 45 for more details on protected libraries.

If the child cell is in a different cell library than the root cell, you may be restricted from saving changes made to it. The EDIT command will prompt you with a warning message if the cell is in a protected library.

[LOCAL_COPY=(TRUE | FALSE)] [VIEW_ONLY=(TRUE | FALSE)]

For most commands, YES is equivalent to TRUE, and NO is equivalent with FALSE.

These parameters are used mainly in command files. You can skip this information if you are not using EDIT in a command file.

If you prefer to avoid having ICED™ display a prompt if the cell you wish to edit is in a protected library, you can use the LOCAL_COPY or VIEW_ONLY keywords. When these keywords are not used in an EDIT command, and the cell is in a protected library, you will see a prompt similar to:

Cell_name already exists in Q:\path_name

Enter option (V=>VIEW only; L=>edit LOCAL copy; C=>CANCEL):

The LOCAL_COPY=TRUE parameter will automatically create a local copy of a cell from a copy edit library. This has the same effect as replying with a <L> to the above prompt. When you terminate ICED™, the local copy of the cell file will be stored in the working directory.

Use the CELL.EDIT.n system macro to test the library class of a cell. See the Command File Programmer's Reference Manual.

If you create a command file that edits cells in different cell library classes, test the class of each cell and use the LOCAL_COPY=TRUE option in the EDIT command only when the cell is in a copy edit library. When this option is used to edit a cell in a direct-edit or view-only cell library it will generate an error and terminate the command file.

Using the LOCAL_COPY=FALSE parameter has no effect on the EDIT command. The warning prompt will be displayed if no VIEW_ONLY=TRUE parameter is present.

If the VIEW_ONLY=TRUE parameter is used, you will automatically be placed in view-only mode. You will not be able to use the EXIT or LEAVE commands

to save changes to the cell. Using this parameter has the same effect as replying with a <V> to the above prompt. It will allow you to view cells in any library (including direct-edit or protected libraries) without a warning prompt.

The **QUIT** command must be used to close a cell opened in view-only mode.

Let us say you are creating a command file that is looking for information in a list of cells (e.g. counting certain types of geometry). The command file will contain a loop with an EDIT command to open each cell. If the EDIT command contains the VIEW_ONLY=TRUE parameter, each cell will open in view-only mode insuring that no cell is accidentally modified.

Using the VIEW_ONLY=FALSE parameter has no effect on the EDIT command. The warning prompt will still be displayed if the cell cannot be directly edited.

CELL=*cell_name*

One of the keywords CELL, SELECT, or NEAR **must** be used with the EDIT command.

If you want to edit a cell by name, use the CELL keyword to specify the name of the child cell.

Example: **EDIT CELL MYCELL**

This command will open the cell MYCELL for editing. The parent cell you were working on is still open, and its cell file is not yet saved. However, all commands will now operate on cell MYCELL and not the parent cell. You must use one of the exit commands (EXIT, QUIT, or LEAVE) to close MYCELL and return to the parent cell.

If MYCELL is a new cell, its cell file will be stored in the working directory. If MYCELL already exists in any of the cell libraries defined with the current project, then the first copy of MYCELL.CEL found by ICED™ will be opened. The search path used to find the cell file will be working directory then each of the cell libraries in the order they are defined. See page 45 for more details.

SELECT

The EDIT SELECT command allows you to edit a selected cell. Only cells can be selected prior to executing this command. If there is more than one cell selected when the EDIT SELECT command is executed, ICED™ will assume you want to edit the selected cell with the smallest area. If no cells are selected when the EDIT SELECT command executes, an embedded SELECT CELL * AT command is generated to allow you to select the cell with the mouse.

NEAR *position*

Using the NEAR keyword allows you to traverse several levels of cell hierarchy at a time. If *position* is a coordinate pair that identifies a point near the edge of a visible component (i.e. not BLANKed or hidden by the DISPLAY CELL_DEPTH command) in a deeply nested cell, EDIT NEAR *position* will allow you to edit the nested cell directly.

The size of the near box is set with the NEAR command.

It is more common to use the mouse to identify *position*. If it is not supplied with the EDIT NEAR command, you can select the deeply nested cell to edit by putting the near box cursor over the edge of a component in the desired cell and clicking the left mouse button.

Example:

EDIT NEAR

Since the *position* parameter is not included in this command, ICED™ expects you to supply it with the mouse. ICED™ will display a near box cursor, similar to the one it displays for a SELECT NEAR command. Position the cursor as if you were selecting a component in the cell you wish to edit. When you press the left mouse button, ICED™ will begin editing the cell that contains the component.

The NOW Keyword and the ENTER.SUBCELL Macro

The following new feature is useful primarily if you use specialized command files and/or macros. This is a subject covered in detail in the Command File Programmers Reference Manual. See that manual for details on macros and command files.

The ENTER.SUB-CELL macro is also executed automatically by the P_EDIT and T_EDIT commands.

The layout editor will now look for the ENTER.SUBCELL macro when an EDIT command is executed. If the macro exists, the string stored in it is executed as a command string. If the macro does not exist, the EDIT command continues without any errors. Add the NOW keyword to the EDIT command to avoid executing the macro.

Example:

GLOBAL #ENTER.SUBCELL=@EDITSUB.CMD

When the command above is executed, it defines an ENTER.SUBCELL macro. This macro is defined with a command string that will execute the EDITSUB.CMD command file. When the macro is executed as a command string, the commands in the file EDITSUB.CMD will be executed as though they were typed at the keyboard.

Example:

EDIT CELL=MYCELL

When the command above is used to edit the cell MYCELL, and the macro definition above has been executed in the current session, then the EDITSUB.CMD command file will be executed after MYCELL is opened, but before the cell is displayed in the view window.

Example:

EDIT NOW CELL=MYCELL

When the NOW keyword is added to the EDIT command, the ENTER.SUBCELL macro is ignored and the EDIT proceeds without executing any additional commands.

The EXIT.ROOT or EXIT.SUB-CELL macros are executed when you leave a cell. See page 199.

EXIT

Terminate the edit session and save the cell.

EXIT [NOW]

Refer to the **JOURNAL**, **LEAVE**, and **QUIT** commands for other methods of terminating an edit session.

ICED™ cannot process an EXIT command if the **VIEW ONLY** mode is set. See page 85.

See page 98 for details on cell file names and cell backup file names.

The EXIT command is one way to terminate editing a cell. The current cell file will be saved when you terminate ICED™ (unless you use the JOURNAL command to terminate the layout editor). When you use EXIT while editing the root cell (the cell edited when you first launched ICED™), you terminate ICED™ and close the editor.

Unlike the LEAVE command, EXIT will cause the cell file to be overwritten even if no changes were made to the cell. This means that when you use ICED™ to browse a cell, then use the EXIT command, the cell file is overwritten and the backup of the cell is lost. For this reason, it is a good idea to routinely use the LEAVE command instead of EXIT.

When EXIT is used to return from a child cell edit session entered using the EDIT, P_EDIT, or T_EDIT commands, the child cell is flagged for saving and you are returned to the parent cell. The child cell file is not saved to disk until you terminate ICED™ completely. If your system crashes, or if the JOURNAL command is used to terminate ICED™, no cell files will be saved.

If you are editing the root cell, EXIT will terminate ICED™. Before ICED™ terminates all flagged cells will be saved. Any existing cell files will be copied to backup files before they are overwritten. Each cell will be saved in a cell file with the name *cell_name.CEL*. ICED™ will also rename the journal file (listing all commands executed during the current session) as *cell_name.LOG*.

See the following table for a comparison of EXIT to the other termination commands.

	EXIT	QUIT	LEAVE	JOURNAL
Primary use	To save changes	For browsing and to avoid saving mistakes	To save cell file only when changes are made	To avoid saving any cell files, but allow for automatic recovery
Is current cell flagged for saving?	Yes	No	Only if geometry has been changed	No
When used to terminate from root cell, are flagged cells saved and journal file renamed before editor closes?	Yes	Yes	Yes	No
Other considerations	Cell files are overwritten and backups of cell files are lost, even if cell was not modified.	Automatic recovery of changes to the cell will not be enabled.	It is a good habit to use LEAVE instead of EXIT, since no backups will be lost when browsing.	Editor closes immediately, and no files are saved, even when editing a child cell.

Figure 65: Comparison of termination commands

The NOW Keyword and the EXIT.ROOT and EXIT.SUBCELL Macros

The following new feature is useful primarily if you use specialized command files and/or macros. This is a subject covered in detail in the Command File Programmers Reference Manual. See that manual for details on macros and command files.

The layout editor will now look for special macros when an EXIT command is executed. If the macro exists, the string stored in it is executed as a command string. If the macro does not exist, the EXIT command continues without any errors. Add the NOW keyword to the EXIT command to avoid executing these macros.

The LEAVE command will also execute one of these macros when the cell is modified.

EXIT.ROOT	automatically executed when EXIT is used to close the root cell and terminate the editor.
EXIT.SUBCELL	automatically executed when EXIT is used to return from a subcell edit.

Example: **GLOBAL #EXIT.ROOT=@CLOSEDOWN.CMD**

When the command above is executed, it defines an EXIT.ROOT macro. This macro is defined with a command string that will execute the CLOSEDOWN.CMD command file. When the macro is executed as a command string, the commands in the file CLOSEDOWN.CMD will be executed as though they were typed at the keyboard.

Example: **EXIT**

When the EXIT command is used to close the layout editor, and the macro definition above has been executed in the current session, then the CLOSEDOWN.CMD command file will be executed before the editor closes.

Example: **EXIT NOW**

The ENTER.SUB-CELL macro is executed when you edit a subcell. See page 196.

When the NOW keyword is added to the EXIT command, the EXIT.ROOT or EXIT.SUBCELL macro is ignored and the EXIT proceeds without executing any additional commands.

FILL

Turn component fill patterns ON or OFF.

FILL [ON | OFF] [ALL | MIXED]

The FILL command is used in conjunction with the LAYER and PATTERN commands to determine how components are displayed on the screen. How a component is displayed depends on:

- the pattern file loaded by the PATTERN command,
- which pattern from the pattern file is assigned to the component's layer by the PATTERN parameter of the LAYER command,
- and
- whether fill has been turned ON or OFF by the FILL command.

When FILL is turned OFF, all components are drawn in outline mode in the colors assigned to their layers. However, components on layers assigned pattern 0 will appear somewhat lighter than those assigned other PATTERN numbers. (ICED™ will only draw every other dot in their outlines.)

When FILL is turned ON, the fill pattern used in drawing a component depends on the pattern number assigned to its layer. Pattern 0 implies no fill (outline mode). Pattern 1 implies solid fill. The other pattern numbers imply the use of user definable fill codes created with the MkSti utility and loaded into ICED™ with the PATTERN command.

The **MkSti** utility is used to create new fill patterns.

Use the PEN parameter of the **LAYER** command to assign patterns for plotting.

When fill is turned ON, text and line components will be drawn as solid white lines. When fill is turned OFF, they are displayed in the color of the layer they are on.

If you are using the P_EDIT (EDIT in place) command, you can use the FILL MIXED command to modify the way fill is applied. In the ON MIXED mode, objects in the cell you are editing, or in its subcells, will be filled. Objects in other cells will be drawn in outline mode. The MIXED parameter is ignored for FILL OFF. The MIXED keyword has no effect outside of P_EDIT.

Example: **FILL**

This form of the FILL command, without any keywords, toggles between the FILL ON and FILL OFF modes.

GRID

Define display grid parameters.

GRID 1 [ON | OFF] [[COLOR=]*color_id*] [[STYLE=](DOTS | LINES | CROSSES)] ...
... [STEP=*real*]
GRID 2 [ON | OFF] [[COLOR=]*color_id*] [[STYLE=](DOTS | LINES | CROSSES)] ...
... [STEP=*integer*]
GRID 3 [ON | OFF] [[COLOR=]*color_id*] [[STYLE=](DOTS | LINES | CROSSES)] ...
... [STEP=*integer*]

The **SNAP** and **RESOLUTION** commands control the grid for entering coordinates.

There are three screen grids available in ICED™ which are displayed as dots, lines, or crosses. Grids are numbered 1, 2, and 3. You can assign a color to each grid and set the distance between tick marks. The grids affect only the display. They do not affect the way the cursor is snapped when entering coordinates.

A grid may be turned ON (made visible) or turned OFF (made invisible). Turning grid 1 off also prevents grids 2 and 3 from being displayed. Turning off grid 2 prevents grid 3 from being displayed. Turning off grid 3 does not affect grids 1 or 2.

Example:

GRID 2 OFF

This command turns off grid 2. It makes both grid 2 and grid 3 invisible.

All grids that have been turned ON will be displayed on the screen simultaneously. Grid 2 is drawn on top of grid 1, and grid 3 is drawn on top of both of the other grids. When the view scale is such that the grid spacing of a specific grid becomes too fine to be of much use, the grid will not display. This does not affect the ON/OFF status of the grid, and the grid will display again automatically when the view is zoomed in. At some view scales, closely spaced line grids may be displayed as dots.

[[COLOR=]*color_id*]

The **COLOR** command assigns color names.

The *color_id* specifies the color of the grid. It may be a color number in the range 0:15 or a previously assigned color name. When a color number is used, the COLOR=keyword must be used, otherwise it is optional.

[[STYLE=](DOTS | LINES | CROSSES)]

The STYLE keyword is optional since the parameter keywords are unambiguous.

If the LINES style is chosen, the lines will be drawn every other pixel in the horizontal direction and every third pixel in the vertical direction.

[STEP=*real*] or [STEP=*integer*]

The STEP parameter defines the distance between tick marks on the grid. The grids are not independent of each other. Grid 1 steps are defined in user units; grid 2 steps are defined as multiples of grid 1 steps; and, grid 3 steps are defined as multiples of grid 2 steps.

Example:

GRID 1 ON STEP=.5 COLOR=RED DOTS
GRID 2 ON STEP=10 COLOR=CYAN CROSSES
GRID 3 ON STEP=10 COLOR=WHITE LINES

Assuming one user unit represents a micron, then these three commands set the grid 1 spacing at 0.5 microns, the grid 2 spacing at 5 microns, and the grid 3 spacing at 50 microns. All three grids will be turned on.

Keyword Order

The order of the parameters is unimportant for the GRID command. All of the GRID keywords are optional. The values for omitted keywords remain unchanged.

Example: **GRID 1**

This form of the GRID command (with no parameters except the grid number) can be used to display the current values of a grid's parameters on the screen.

GROUP and UNGROUP *Group selected components into a cell.*

GROUP "*cell_name*" [YES | NO] [[AT] (*origin_pos* / *)]
UNGROUP

The GROUP command collects fully selected components into a cell. These components are deleted from the current cell, and the newly formed cell is added in their place. Partially selected components are not affected.

The UNGROUP command replaces selected cells and arrays by the components they contain. Only cells and arrays may be selected when you execute the UNGROUP command. If other components are selected ICED™ will report an error.

"*cell_name*"

See page 96 for restrictions on cell file names.

The *cell_name* parameter must be the first parameter in the GROUP command. It will be used as the name of the newly created cell. The quotes around *cell_name* are optional.

When you exit the ICED™ session, the cell file will be created in the working directory.

Example:

GROUP MYCELL

This use of the GROUP command will remove all fully selected components from the current cell and group them in a new cell with the name MYCELL. Since no *origin_pos* parameter is supplied (see below), the mouse must be used to define the new cell's origin. The command will fail if a cell with the same name is already loaded in RAM. If the cell MYCELL already exists, but is not loaded into RAM, you will be prompted before it is overwritten.

[YES | NO]

If a cell with the name *cell_name* already exists in the working directory or in one of the cell libraries defined in the project, but the cell not been loaded into RAM, ICED™ will warn you and ask you if you want to continue. You can avoid having to respond with a <Y> or <N> keystroke by specifying the optional YES or NO keyword in the GROUP command. This parameter is most often used in command files.

If you use the YES keyword in the GROUP command, you will not be warned if an old cell *cell_name* will be overwritten. The old cell file will be overwritten when you exit the editor.

[AT] *origin_pos* or *

origin_pos is the coordinate pair which defines the origin of the new cell. You can also specify the origin with an asterisk (*). In this case ICED™ will use the lower left hand corner of the cell's bounding box as its origin. If this point is not on the resolution grid, ICED™ will round its coordinates **up** to the nearest resolution grid point.

See the
RESOLUTION
command to
learn about the
resolution grid.

If you do not provide either the *origin_pos* or an * as part of the GROUP command, you must use the mouse to define the new cell's origin. This origin can be outside of the new cell's bounding box.

Examples:

GROUP NEWCELL AT (15.2, 30.0)
GROUP NEWCELL2 *
GROUP NEWCELL3

The first example will delete all fully selected components from the current cell and use them to create a new cell with the name NEWCELL. The origin of the new cell will be at the coordinates (15.2, 30.0) of the current cell.

The second example will create a cell NEWCELL2. The origin of this cell will be the lower left hand corner of the rectangle containing all of the selected components. Note that the AT keyword is optional. For the third example, you will be prompted to define the origin of NEWCELL3 with the mouse.

UNGROUP

The UNGROUP command will delete all selected cells and arrays, and will replace them with their components. The components can then be manipulated individually just like any other components in the current cell. If components other than cells and arrays are selected when the UNGROUP command is executed, ICED™ will report an error.

See the **EDIT**,
P_EDIT, and
T_EDIT
commands to
modify subcells.

The UNGROUP command will not affect the saved cells files. It will also not unload the cell from RAM. Therefore, you cannot UNGROUP a cell, edit it, then use GROUP to recreate a cell with the same name. You should use one of the EDIT commands to modify the cell instead.

Interactions Between the GROUP/UNGROUP and SELECT Commands

The easiest way
to copy a set of
components
into another cell
is to **GROUP**
the components
into a cell, **ADD**
the new cell to
another cell,
then
UNGROUP the
new cell.

The components you wish to group together in a cell must be selected prior to executing the GROUP command. If you try to issue a GROUP command when no components are selected, ICED™ will report an error.

If you issue a SELECT NEW command just after a GROUP command, ICED™ will select the newly grouped cell.

If you issue an UNGROUP command and no components are selected, ICED™ will automatically generate an embedded SELECT CELL * AT command. You then use the mouse to select a cell by clicking the left button when the cursor is inside the boundaries of the cell.

If you issue a SELECT NEW command just after you UNGROUP a cell or ARRAY, ICED™ will select newly ungrouped components.

Note on the UNDO Command

The UNDO command **cannot** be used to reverse the effects of the GROUP command. However, the effects of the UNGROUP command can be reversed with the UNDO command.

INITIALIZE

Set the parameters of a list of layers.

INITIALIZE LAYERS=*layer_list*

See the
LAYER
command.

INITIALIZE is used to remove any unique parameters (e.g. layer names) from a list of layers. Using this command will reset the parameters of the layers in *layer_list* to the values applied to layer *, the fictitious layer used to define the defaults for all unnamed layers. (The LAYER * command is used to set the parameters of layer *.) INITIALIZE applies those parameters to all layers in *layer_list* as well. This also has the effect of clearing the names from all layers in *layer_list*.

See the section
on **Layer Lists**
in the **Syntax**
Conventions
chapter for
details on how
to use layer
lists.

The LAYER * command is usually executed by the startup command file run on all new cells. If for some reason a LAYER * command has not been executed, the defaults are as follows:

```
LAYER * NO_NAME WIDTH=4.0 COLOR=YELLOW ...  
... PAT=1 NO_PEN NO_CIF NO_STREAM
```

The *layer_list* parameter for the INITIALIZE command defaults to 0:255 (i.e. all layers).

Example:

INITIALIZE LAYERS=1:*

This example will apply the parameters set with the LAYER * command to all layers except for layer 0. If any of those layers was previously named, the name is removed along with any other unique layer parameters such as color or patterns.

The INITIALIZE command is usually used in the startup command file immediately after the LAYER * command. While LAYER * will not affect named layers, pairing it with an INITIALIZE *.* command insures that all layers are unnamed and have the same parameters.

JOURNAL

Terminate ICED™ without saving work.

JOURNAL

Refer to the **EXIT** command for a comparison of termination commands.

The JOURNAL command is used to terminate ICED™ without saving any changes. No cell files are saved and none of the changes made during the current session are kept. Any changes you made to other cells using the EDIT, P_EDIT, or T_EDIT commands are also lost. This is true even if you have already exited these cells since no cell files are saved until you terminate ICED™ using one of the other termination commands.

Unlike the EXIT, LEAVE, and QUIT commands, JOURNAL will close the editor even if you are in a nested edit session.

See **Journaling and Data Recovery** in **Appendix B**.

The JOURNAL command does preserve the journal file containing a record of all commands executed during the editing session. This file has the same name as the root cell with a .JOU extension. It can be used to automatically recover changes made to the cells.

If you launched ICED™ to edit the cell OPAMP, the journal file will be created with the name OPAMP.JOU. If you terminate the program with the JOURNAL command, no cell files will be saved and the journal file will not be renamed.

The next time you open the editor to edit opamp, ICED™ will prompt you:

Journal file for OPAMP exists
Do you want to recover [Yes, No, or Quit]?

Responding with a <Y> will execute all commands entered during your last unsaved edit session. If you used the JOURNAL command because you made errors in the last ICED™ session and you wish to recover the cell as it was **before** that session, respond with a <N> to the above prompt. The journal file for the previous session is renamed with a .JOX extension as soon as you respond <N>

but it can still be used for recovery at a later time by executing it explicitly as a command file. Typing a <Q> will prevent the editor from launching and will preserve the journal file, but it has already been renamed with a .JO1 extension. The file is still valid for automatic recovery.

You can edit the journal file before launching ICED™ again and responding with a <Y> to the recovery prompt. This will give you the chance to recover part of your work. You should read **Journaling and Data Recovery** in **Appendix B** before attempting this type of recovery.

KEY

Assign a command string macro to a function key.

Key (*F_n* | *AF_n* | *CF_n* | *SF_n*) = "*cmd_string*"

The KEY command is used to assign command string macros to function keys. Function key assignments allow you to execute a complicated series of commands with a single keystroke.

KEY commands create a special case of an ICED™ macro. In general, macros let you replace a string or keystroke with another string or equation. The macro definition created by a KEY command will look similar to:

GLOBAL #KEY.F1="*cmd_string*"

There are four different quote characters available in ICED™: ", ', ~, and `. If *cmd_string* contains one of these characters, a different quote character must be used to surround it. For example, if the command string contains a double quote (") you must surround it with single quotes (') or tildes (~) or accent marks ('). There is no way to create a string containing all four quote characters.

The use of quotes is usually optional when typing the KEY command. Quotes must be used to delimit the *cmd_string* if it contains characters that may confuse the parser such as the '@', ';', '!', or '&' characters, or quote characters.

Some users have reported problems with the <F10> key. This appears to be an operating system limitation.

Using *F_n* in the KEY command (where *n*=1, 2, ... or 12) assigns *cmd_string* to <*F_n*>, one of the keyboard keys labeled F1 - F12. Use the letter F and a number in the KEY command definition and later use <*F_n*> to execute *cmd_string*.

Similarly, using *SF_n* in the KEY command assigns *cmd_string* to the keyboard combination <SHIFT> <*F_n*>. Use the letters SF in the KEY command definition and later hold down the <SHIFT> key and type the <*F_n*> key to execute *cmd_string*.

AF_n assigns commands to the <ALT> <*F_n*> key combination. *CF_n* assigns commands to the <CTRL> <*F_n*> key combination.

Example: **KEY F1="RULER"**

After executing this command it is possible to execute the RULER command by typing <F1>.

Example: **KEY SF9="@D:\ICED\AUXIL\PLOTALL"**

See
@*file_name* for
details on
command files.

Using this KEY command definition allows you to execute the indicated command file by holding down the <SHIFT> key and typing the <F9> key. It is possible to execute very complex lists of commands with a single keystroke.

Example: **KEY F2='ADD TEXT "GND" LAYER M1 SIZE 4'**

Notice that single quotes (') were used to delimit the macro since the ADD TEXT command included double quotes (").

Saving your Macros

The
TEMPLATE
command can
also save KEY
assignments in a
file.

Key macros are recorded in the cell files with all the other cell information. Thus, if you assign a command to a function key during one editing session, ICED™ will remember the key assignment the next time you edit the same cell. However, you will probably wish to use many of the same keyboard macros in all your cells. You can do this by using the SHOW command to create a command file that contains your keyboard macros.

Example: **SHOW USER=KEY.* FILE=MACRO.CMD**

See page 25 for
details on
startup
command files.

This command will create a file MACRO.CMD that contains the current KEY assignments. You may add the key assignment commands in MACRO.CMD to your startup command file with one of two methods. One way is to use an ASCII text editor to copy the key assignments into the startup command file. Another way is to add the line @MACRO.CMD to the startup command file. The second method will cause the startup command file to execute all commands in the file MACRO.CMD.

To add saved key assignments to an existing cell without modifying the startup command file, simply execute @MACRO.CMD in the existing cell.

Limitations

You **cannot** use a function key for string replacement. For example, if you typed **KEY F1="RED"**, then later typed **LAYER 1 COLOR=<F1>**, the <F1> key would **not** fill in the command with the string RED.

LAYER

Set the default layer or layer parameters.

Syntax for setting the default layer:

LAYER =*layer-id*

Syntax for setting layer parameters:

LAYER (*layer-id* | *) ...
 ... [NAME=*name* | NAME="" | NO_NAME] ...
 ... [WIDTH=*def_wire_width*] ...
 ... [SPACE=*def_spacing_distance*] ...
 ... [[COLOR=]*color_id*] ...
 ... [PATTERN=*pattern_no* | SOLID | DOTTED] ...
 ... [PEN=*pen_no* | PEN=* | NO_PEN] ...
 ... [CIF=*cif_name* | CIF="" | NO_CIF] ...
 ... [STREAM=*stream_layer_no* [, *stream_data_type* [, *stream_text_type*]] | NO_STREAM]

The LAYER command is used to set the default layer used by the ADD command and to set layer characteristics.

Setting the Default Layer

The **USE LAYER** command can also set the default layer.

The default layer is the layer to which components (other than cells and arrays) are added if no layer is specified in an ADD command. The default layer can be set by using the first form of the LAYER command indicated above. For this use of the layer command, the *layer-id* can be any integer from 1:255 or any previously defined layer name.

Examples: **LAYER 55**
 LAYER POLY

The first command sets the default layer to layer 55. The second command sets it to a layer named POLY. If the name POLY has not been assigned to a layer with a previous LAYER command, ICED™ will report an error.

When setting the default layer with the LAYER command, ICED™ will report the current layer parameters for the specified layer.

Setting the Layer Parameters

The **TEMPLATE** command will report the current parameters for all named layers.

See page 25 for details on startup command files.

ICED™ supports 255 layers numbered 1 to 255²². Each layer can have a separate set of parameters assigned to it. In addition, there is a fictitious layer 0 used to display the outlines of cells and arrays. ICED™ generates these outlines automatically. You can not add any components to layer 0. The only parameter you can assign to layer 0 is the pen number.

The layer parameters are most commonly set by a series of LAYER commands in the startup command file that ICED™ executes when you create a new cell. The layer parameters are saved in the environment database of the cell file (except for the CIF and STREAM parameters as explained later).

If you edit a startup command file and change the layer parameters, those new parameters are loaded only in **new** cell files. Old cell files will still use the obsolete layer parameters. To update the layer parameters of a few old cell files, you can execute the new startup command file with the @* command while editing the old cell files. Be sure to save each old cell file with the EXIT command.

If you change layer number-layer name correspondences, this may lead to hazardous inconsistencies in your cell libraries. If you need to make changes to the layer names in the startup command file, you should then reinitialize the layer number-layer name correspondences in all cell files. This can be accomplished for all cells in a given design by exporting the highest level cell

²² Layers 250 : 255 are used for scratch work by IC EDITORS, Inc. and other writers of command files. Therefore, you should avoid creating maskable geometry on these layers.

with a STREAM ARCHIVE command, then recreating all of the cells with the UnStream import utility.

In examining the example commands, please remember that ICED™ generally treats the characters '=', ',', '(', and ')' as if they were blanks. These characters are included in ICED™ commands to make them more readable but usually serve no other purpose. Also, all kinds of names, including layer names and CIF layer names, can be enclosed in quotes. These quotes are generally optional. The only exception is in specifying null names which can be entered as "".

Use of Layer Names, Numbers, and *

Combine the **LAYER *** command with the **INITIALIZE** command to initialize both named and unnamed layers.

A LAYER command can set the parameters for a single layer or for all unnamed layers. A *layer_id* specifies a single layer whose parameters you wish to change. It may be a layer number or a previously assigned layer name.

An asterisk (*) indicates that the command should set parameters for all unnamed layers (except for layer 0). A LAYER *... command will assign the parameters indicated in the command to a fictitious layer which defines the defaults for all unnamed layers. Then the command goes on to assign those same parameters to all unnamed layers.

Keyword Order and Conflicts

When used to set layer parameters, a LAYER command always starts with the keyword LAYER followed by *layer_id* or * followed by one or more additional keywords or parameters. The additional keywords are all optional and their order is unimportant.

If *layer_id* is 0, only the PEN parameter can be set. If *layer_id* is replaced by a *, only the WIDTH, COLOR, PATTERN, PEN, NO_CIF, or NO_STREAM parameters can be set.

[NAME=*layer_name* | NAME="" | NO_NAME]

Layer names are stored in a cell's environment database.

Layers 1 to 255 can be named. Once named, a layer can be referred to by either its layer name or layer number, however, it is the layer number that is stored with the component. Unnamed layers can only be referred to by their layer numbers. All layer names must be unique and no more than 8 characters long. Layer names may include letters, numbers, or the special characters '\$', '_', and '#'. Layer names may **not** start with a number.

Example: **LAYER 3 NAME=POLY**
 LAYER POLY NAME "PWEL"

In this example, layer 3 is named POLY and then renamed PWEL. Notice that the '=' and the quotes around the layer name are optional.

You can unname a layer by using the NO_NAME keyword or by setting its name to an empty string (entered as ""). The following three commands have the same effect.

Example: **LAYER POLY NO_NAME**
 LAYER POLY NAME=""
 LAYER POLY NAME=" "

The quotes were required in the last two examples.

The NAME parameter cannot be set using a 'LAYER *' command.

[WIDTH=*def_wire_width*]

See the **NDIV** command line parameter for more information on database units. By default, NDIV=1000.

The WIDTH parameter determines the default wire width used by the ADD WIRE and ADD ARC commands. It also determines the default text size used by the ADD TEXT command. The *def_wire_width* parameter is measured in user units. It must be a positive real number such that:

$$def_wire_width * NDIV = \text{an even integer}$$

In other words, one half of the *def_wire_width* must be an integral number of database units.

[**SPACE**=*def_spacing_distance*]

See more details on the spacing cursor in the description of the **SPACER** command.

This parameter is used to control the appearance of the spacing cursor. It will not prevent you from violating any minimum spacing requirements. It merely controls a visual aid that helps you avoid violating spacing rules.

The spacing cursor can be used during most ADD commands. It adds guide marks a certain distance away from the center of the cursor to help you place a new component a certain minimum distance away from existing components. One of the ways to set this minimum distance is to use the existing **SPACE**=*def_spacing_distance* of the current default layer.

For the spacing cursor to automatically change the spacing distance when the default layer is changed, the spacing cursor must be turned on and have the TRACK_LAYERS mode turned on. Then, if all of your layers are defined with valid **SPACE**=*def_spacing_distance* parameters, every time you change the default layer, the correct minimum space for that layer will be used to draw the spacing cursor.

The *def_spacing_distance* always defaults to 0 for any layer defined without a valid **SPACE** parameter. Set *def_spacing_distance* for each layer to an appropriate positive non-zero real number of user units.

[[**COLOR**=]*color_id*]

Color names and parameters are set by the **COLOR** command.

The **COLOR** parameter determines the color in which the components on a given layer will be displayed. The *color_id* may be a color number in the range 1:15 or a previously assigned color name. (The color names for colors 1 through 7 are WHITE, YELLOW, GREEN, RED, CYAN, BLUE, and MAGENTA. The color names for colors 8 through 15 are assigned using the **COLOR** command.)

Example: **LAYER 1 COLOR=RED**
 LAYER 1 COLOR=4
 LAYER 1 "RED"
 LAYER 1 RED

These four commands all have the same effect. The last form of the command can only be used in cases where the color name cannot be confused with a keyword.

Example: **LAYER 1 NAME="PWEEL" COLOR=%**

This command will cause ICED™ to prompt you for a color name or number.

[**PATTERN**=*pattern_no* | **SOLID** | **DOTTED**]

Use the **FILL** command to enable display of patterns set with this keyword.

ICED™ can display geometries using solid fill, no fill (outlines only), or user definable stipple patterns. The definitions of the stipple patterns are stored in a stipple pattern file (.STI file). The MkSti utility is used to create the .STI file, the **PATTERN** command must be used to load it into memory, and the **PATTERN** parameter set by the **LAYER** command is used to assign specific patterns to individual layers for screen display. The **FILL** command is used to enable or disable pattern fill for screen display. (Use the **PEN** parameter of the **LAYER** command (covered later) to assign patterns for plotting.)

See the **PATTERN** command for the ready-to-use patterns provided with ICED™.

If **FILL** is turned on, components will be drawn with their assigned fill patterns. If **FILL** is turned off, all components will be drawn without fill (outlines only).

The *pattern_no* can be any integer in the range 0:100. Pattern 0 implies no fill. Pattern 1 implies solid fill. All other patterns must be defined in the .STI file and loaded with the **PATTERN** command. If a layer is assigned an undefined pattern number, components on that layer are drawn with no fill.

Example: **LAYER 3 RED PATTERN=5**

or, more briefly:

LA 3 R PA=5

Either form of the above command sets the color and pattern number for layer 3.

The following command sets the color and pattern number for all unnamed layers:

Example: **LAYER * PAT=3 YELLOW**

The SOLID and DOTTED keywords are preserved from older versions of ICED™. SOLID means exactly the same thing as PATTERN=1. DOTTED means PATTERN=0. Thus, the following two commands have the same effect.

Example: **LAYER 1 PATTERN 0**
 LAYER 1 DOTTED

[PEN=*pen_no* | PEN = * | NO_PEN]

The PEN parameter is used by the PLOT command. The PEN keyword allows you to associate a fill pattern for plotting with each layer.

If you want cell outlines (i.e. dotted lines drawn on the boundaries of cells and arrays) to display in your plots, assign a non-negative pen number to layer 0.

(In earlier versions of ICED™, the PEN keyword was used to select a pen color for pen plotters. That is why it is named PEN. When raster printers became popular, this keyword performed both pen selection for pen plotters and pattern selection for raster printers. Now that pen plotters are no longer popular, this keyword is used only for plot pattern selection. Changing the keyword name would affect too many users adversely. Now you know why the keyword is named PEN.)

This keyword is very important when plotting or generating bitmaps with the PLOT command. It is often the case that patterns that look good on the screen are too dense or too faint on plots due to different resolutions. For example, layers that use pattern 1 (solid fill) on the screen often use a different pattern for plotting since solid fill of large shapes on plots uses a lot of ink and can deform the paper.

However, if you prefer to use the same patterns for plotting that you do on the screen, see the Importance of PEN=* in Plots on page 224.

When PEN=*pen_no* is used, *pen_no* refers to a pattern number in a .STI file. PEN=0 indicates that the only the outline is drawn on the plot. PEN=1 indicates that all geometry on the layer will be plotted with a solid fill. **You should avoid using PEN=1 for layers with large shapes if you are using an Inkjet or LaserJet printer for plotting.** For these types of printers, large areas of solid fill can deposit too much ink on the paper and deform it.

Refer to the **MkSti** utility to create new patterns.

Other PEN numbers will cause component outlines to be filled with patterns from the pattern file (often referred to as the .STI file) specified with the PATTERN keyword on the PLOT command line.

The pen number does not have to be unique. More than one layer can be assigned the same pen number. The layer command will accept pen numbers in the range from -1:100.

The following examples assign plot pattern 5 to layer POLY and plot pattern 6 to layer 10.

Examples: **LAYER POLY PEN 5**
 LAYER 10 PEN=6

The PLOT command will plot only layers that have been assigned non-negative pen numbers. You can remove a pen assignment using the NO_PEN keyword or PEN=-1. The following two commands are equivalent. Both assign NO_PEN to all unnamed layers. This prevents them from being plotted.

Example: **LAYER * PEN=-1**
 LAYER * NO_PEN

Layer 0 is the layer used to represent cell outlines. If layer 0 is assigned NO_PEN (or PEN=-1), then no cell outlines or labels will be drawn on a plot. Assign PEN=0 to layer 0 if you do want cell outlines drawn on your plot.

Example: **LAYER 0 PEN=0**

Importance of PEN=* in Plots

PEN=-2 is an
obsolete way of
assigning
PEN=*,

When PEN=* is used in a LAYER command, the same pattern number defined with the PATTERN keyword (for screen display) is used for plots as well.

This is a good initial default to use so that the plot patterns will match the screen patterns. When you are experimenting with colors and patterns in your layer definitions, it is a common mistake to change only the screen pattern number and to forget to change the plot pattern number.

Example: **LAYER 1 NAME=PWEL WIDTH=3.0 BLUE PAT=0 PEN=***

Let us assume that as you are experimenting with layer definitions to achieve a set that make your particular technology and design easy to read. The command above is used as a first attempt to define a well layer. However, you later determine that an unfilled, blue outline is not enough to see your well shapes clearly. You experiment with other color and pattern choices to make the layer easier to see. You may try several combinations, changing the screen pattern each time with the PATTERN keyword (abbreviated to PAT in the example above). When you finally decide on a lighter blue color with a sparse pattern using the command below, the pattern change applies to plots as well, since the layer was initially defined with PEN=*. If the layer had instead initially been defined with PEN=0, that would remain the plot pattern despite the new definition.

Example: **LAYER PWEL DIM_BLUE PAT=36**

You will probably have at least a few layers that look best with different plot and screen patterns. However, it is easier to initially define them with the same pattern by using PEN=* until you are refining only the plot definitions.

[**CIF=cif_name** | **CIF=""** | **NO_CIF**]

See the **CIF** command for details on CIF file creation.

The CIF parameter is used by the CIF command in making Caltech Intermediate Form output files. In CIF format, layers are referred to by their CIF layer names. The CIF keyword allows you to associate a CIF layer name with each ICED™ layer. *cif_name* can consist of up to 4 letters and digits. *cif_name* does not have to be unique, that is, multiple layers can have the same CIF layer name. A zero length *cif_name* (entered as "") will prevent components on that layer from being included in the CIF file.

Example: **LAYER 1 CIF=WELL**

This command assigns the CIF layer name WELL to ICED™ layer 1.

Example: **LAYER "NDIF" CIF="ACTV"**
 LAYER PDIF CIF=ACTV

The two commands above map both ICED™ layers NDIF and PDIF onto the CIF layer ACTV. Notice that the quotation marks around both the ICED™ and CIF layer names are optional.

The CIF command only outputs layers that have been assigned valid CIF layer names. Any existing CIF layer name can be cleared using the NO_CIF keyword or by entering the *cif_name* as a pair of quotation marks.

Example: **LAYER 3 NO_CIF**
 LAYER 3 NOCIF
 LAYER 3 CIF=""
 LAYER 3 CIF=" "

These four commands are equivalent. They all clear the CIF layer name for layer 3. This means that components on layer 3 will not be output by the CIF command.

Example: **LAYER * NO_CIF**

This command clears any CIF names that might have been assigned to unnamed ICED™ layers.

Please also read "A Special Note on CIF and STREAM" on page 227.

[STREAM=*stream_layer_no* [, *stream_data_type* [, *stream_text_type*]] | NO_STREAM]

The MAP_LAYERS keyword **must** be used in the **STREAM** command to use the STREAM parameters set with the LAYER command.

The STREAM parameter is used by the STREAM command in making Stream (Calma-GDSII) compatible output files. In stream format, each component (except for cells and arrays) has both a layer number and type number. Maskable components (i.e. polygons, boxes, and wires) and lines will be assigned a data type number. Text components will have a text type number instead of a data type number. Data types and text types may be used by other programs (such as design rule checkers available from other vendors). The STREAM parameter allows you to associate a stream layer number, stream data type, and stream text type with each ICED™ layer.

Any integer in the range -1 to 255 can be used for *str_layer_no*. The value -1 prevents components on that layer from being included in the Stream file. *str_data_type* and *str_text_type* can be any integers in the range 0 to 255. The strict Calma standard allows only integers in the range 0:63 for the Stream layer number, data type, or text type. Most modern software reading Stream files supports 255 layers, but you should be aware that layers or types in the range 64:255 may cause problems for some programs.

A layer's stream layer number and stream data or text type do not have to be unique. That is, multiple layers can have the same stream layer number, stream data type, or stream text type.

Example:

```
LAYER 10 NAME M1    STREAM 5, 0, 3
LAYER 11 NAME M1G  STREAM 5, 1, 4
LAYER 12 NAME M1V  STREAM 5, 2, 5
LAYER 13 NAME M2    STREAM 63, 0, 3
LAYER 14 NAME M2G  STREAM 63, 1, 4
LAYER 15 NAME M2V  STREAM 63, 2, 5
```

In this example, six ICED™ layers are mapped into two stream layers each with three different data types and three different text types. Maskable components on layer M1 will have be assigned to stream layer 5 and data type 0. Text components on M1 will also be assigned to stream layer 5, but they will have text type 3.

If a LAYER command includes the STREAM keyword, but does not include a value for *str_data_type*, the stream data type is set to 0. Any preexisting value for the stream data type is lost. *str_text_type* also defaults to 0.

Example: **LAYER 11 NAME=M1 STREAM 1, 0, 0**
LAYER 11 NAME=M1 STREAM 1, 0
LAYER 11 NAME=M1 STREAM 1

All three commands are equivalent. They all define the stream layer number associated with layer 11. The data type and text type are both set to 0 in all three examples.

The STREAM command will output only layers that have been assigned valid stream layer numbers. You can clear any previous stream layer assignments for a layer by setting its stream layer number to -1 or by using the NO_STREAM keyword.

Example: **LAYER 1 NO_STREAM**
LAYER 1 NOSTR
LAYER 1 STREAM=-1

All three commands are all equivalent. They all undefine the stream layer number associated with layer 1. This means that components on layer 1 will not be output by the STREAM command (if the MAP_LAYERS keyword is used).

Example: **LAYER * NO_STREAM**

This command clears any stream layer numbers that might have been assigned to unnamed ICED™ layers.

A Special Note on CIF and STREAM

Usually, all the parameters you set while working on a cell are saved in its cell file when you exit ICED™. However, ICED™ deliberately forgets the CIF and STREAM parameters when you exit a drawing. If you edit an old cell and do not redefine CIF parameters before executing a CIF command you will get the error message: "No layers assigned CIF layer names."

Similarly, if you do not redefine the STREAM parameters before executing a STREAM command you will get the error message: "No layers assigned STREAM layer numbers." The CIF and STREAM parameters are normally defined in your startup command file. You can execute the startup command file by typing @* <Enter> or choosing @START from the third menu.

This behavior was incorporated in the early days of ICED-16 after several near disasters. A user would change the CIF layer names in his startup command file. Later, he would edit an old drawing and execute the CIF command. Since ICED™ does not normally execute the startup command file when editing old cells, it was unaware of the new CIF layer names. The existence of obsolete layer correspondences could lead to an incorrect mask set, a very costly mistake. ICED™'s current behavior forces you to reload the CIF and STREAM parameters using an up to date command file.

LEAVE

Terminate the edit session and save the cell if it has changed.

LEAVE [NOW]

Refer to the **EXIT** command for a comparison of termination commands.

The LEAVE command is used to terminate editing the current cell. Unlike the EXIT command, LEAVE will **not** save the current cell if no changes have been made to the geometry in the cell.

The LEAVE command flags a cell for saving only if components have been modified. If no changes were made, or if only environment database changes were made (i.e. changes to fill patterns or colors), the cell file will not be flagged for saving.

See **How ICED™ Saves Data** for more details on cell files and backups and how they are saved.

When you use LEAVE to return from editing a child cell edited using the EDIT, P_EDIT, or T_EDIT commands, you will be returned to the parent cell. If the child cell has been flagged, the cell file will be saved when you terminate ICED™.

If you are editing the root cell (the cell edited when you first launched ICED™), LEAVE will save the root cell file before terminating only if the geometry in it has changed. If other cells were flagged for saving during the current session, their cell files are also saved at this time. LEAVE will then rename the journal file (listing all commands executed during the current session) to *cell_name.LOG*.

See **Journaling and Data Recovery** for details on cell file backups and recovery.

It is a good habit to routinely use LEAVE instead of the EXIT command. When you are only browsing a cell, then use EXIT to terminate, the cell will be flagged and its file will be overwritten, despite the fact that it has not been modified. When the cell file is saved, the cell file backup is also overwritten, and you lose the previous version of the cell. Using LEAVE instead will always preserve one backup of the cell.

The LEAVE command is especially useful in command files since it will not overwrite a cell unless it was modified by the commands in the command file.

Example: **EDIT GEORGE**
 UNSELECT ALL; XSELECT OFF
 SEL LAYER 100 ALL
 DELETE
 LEAVE

See the
 @*file_name*
 command for
 details on how
 to use command
 files.

If these commands were in a command file, the cell GEORGE would be flagged for saving only if there were components on layer 100. If the DELETE command did nothing, the cell would not be saved, the time date stamp for GEORGE.CEL would not be altered, and the contents of the cell file backup (GEORGE.CL1) would not be lost.

The NOW Keyword and the EXIT.ROOT and EXIT.SUBCELL Macros

The following new feature is useful primarily if you use specialized command files and/or macros. This is a subject covered in detail in the Command File Programmers Reference Manual. See that manual for details on macros and command files.

When a LEAVE command is executed, and the cell is flagged for saving because the geometry has changed, the layout editor will now trigger special macros. If the appropriate macro from the list below exists, the string stored in it is executed as a command string. If the macro does not exist, the LEAVE command continues without any errors.

The EXIT
 command will
 also execute one
 of these macros
 if it exists.

EXIT.ROOT	automatically executed when EXIT is used to close the root cell and terminate the editor.
EXIT.SUBCELL	automatically executed when EXIT is used to return from a subcell edit.

If you want to avoid the execution of these macros, add the NOW keyword to the LEAVE command. See the EXIT command for more details.

LIST

Save a named list of components.

LIST [DEFAULT] (GLOBAL | LOCAL) [#]*list_name*

LIST #*list_name* (This syntax is valid only when reusing an existing list.)

This command is primarily of interest to writers of command files. It is used to save a named set of components, allowing you to unselect everything and then select one component at a time from the list with a SELECT LIST command.

(If you will be using LIST in a command file, you may want to read the additional information in the description of this command in the Command File Programmers Reference Manual. Details are also provided in the Q:\ICWIN\DOC\LIST.TXT file.)

The easiest way to copy a set of components into another cell is to **GROUP** the components into a cell, **ADD** the new cell to another cell, then **UNGROUP** the new cell.

Lists are valid only in the cell in which they were defined. Therefore you cannot use a list to copy a set of components to another cell.

The DEFAULT keyword is used to force the editor to ignore a redefinition of an existing list. It has no real purpose outside of a command file.

The LOCAL or GLOBAL keywords define the scope of the list in the same way these keywords work in a macro definition. LOCAL lists are valid only in the command file that defines them, while GLOBAL lists persist until you exit the current cell. (See **When Lists Are Removed** at the end of this command description for more details on how long lists persist.) Outside of a command file, always use the GLOBAL keyword in a list definition.

Example:

LIST GLOBAL #MYLIST

This example defines a list with the name MYLIST. All selected components will be added to the list. These components remain selected after the command.

Example:

```

UNSELECT ALL
ADD CELL NAND
SELECT NEW
UNGROUP
SELECT NEW
LIST GLOBAL #MYLIST
UNSELECT ALL
SELECT LIST #MYLIST NEXT
    
```

SELECT LIST commands can contain the keywords, PREVIOUS, FIRST, or LAST instead of NEXT.

Hold the <Shift> key down during the embedded SELECT NEAR to select multiple components.

To see another example of a SELECT LIST command, refer to page 306 in the **SELECT** command description.

Learn more about macros in the Command File Programmers Reference Manual.

The example above demonstrates the LIST and SELECT LIST commands. The cell NAND is added to the current cell, then ungrouped into its components. Only those components are selected and placed in the list MYLIST. Then all components are unselected, and the first component in the list is selected by the last command. You can repeat the last two commands over and over to select each component imported from the NAND cell one at a time.

It is best to select the components for a list prior to executing the LIST command. (When no components are selected when a LIST command is executed, an embedded SELECT NEAR command is executed to allow you to select components for the list.) Both fully and partially selected components will be put on the list. When a component that was partially selected before being put on the list is selected, it will be fully selected by a SELECT LIST command.

You cannot have more than 6 lists for a given cell at any one time and a single list cannot contain more than 10,000,000 components.

The following restrictions are enforced for a *list_name*:

- List names can be from 1 to 16 characters long.
- Valid characters consist of all alphanumeric characters and '!', '_', and '\$'.
- Do not use a number as the first character in a list name.

The '#' before the *list_name* is optional when defining a list for the first time. This character is often used to identify macro names in many other commands, and a list is simply a special case of a macro. However, since the parser expects a *list_name* after the required LOCAL or GLOBAL keyword, the '#' is not required.

However, if you are redefining the contents of an existing list, you can omit the LOCAL or GLOBAL keywords. In this case, the '#' is required to tell the parser that the following string represents a list name. It is a good programming practice to always precede a list name with a '#'.

Example: **SELECT ALL**
 LIST #MYLIST

The LIST command above will result in a syntax error unless the list MYLIST was previously defined with either the LOCAL or GLOBAL keywords. If this is the case, the previous contents of the list are replaced with the list of all components in the current cell.

When Lists Are Removed

Refer to the
 REMOVE
 command in the
 Command File
 Programmers
 Reference
 Manual.

Local lists are removed automatically when the command file in which they are declared is complete.

Global lists are removed automatically as soon as you EXIT, QUIT, or LEAVE the current cell.

Lists can be deleted at any time with a REMOVE LIST *#list_name* command.

LOG

Speed command files by controlling how commands are logged.

LOG OFF

or

LOG [SCREEN=(ON | OFF)] [LEVEL=(BRIEF | NORMAL | DEBUG)]

See the *@file_name* command for details on using command files.

The LOG command is used to control how commands in command files are logged (i.e. copied) to the journal file and echoed on the status line of the layout editor window. The primary use of these commands is to speed command file execution time.

The journal file is not only a record of what was done during the edit session, but it can also be used to recover from errors in the command file. To learn more about journal files, see page 99.

When no LOG command overrides the default behavior, each of the commands in a command file will be echoed on the screen and logged in the journal file. In many command files, most the execution time is spent performing these command logs.

If you use the LOG OFF command as the first command in a command file, only the *@file_name* command will be logged in the journal file, and no commands will be echoed on the screen. When the LOG OFF command is used, it **must** be the first command executed in the command file. It is an error to execute a LOG OFF command outside of a command file. See **Effects of the LOG OFF Command** below to better understand how this command affects your ability to recover from errors.

Refer to the **VIEW (ON | OFF)** command to speed command files by not updating the display of the cell.

The SCREEN keyword is used to control whether or not commands in the command file will be echoed on the screen display. See page 237. The LEVEL keyword is used to control what kind of commands will be logged. We will cover the different settings on page 238.

LOG [SCREEN=(ON | OFF)] [LEVEL=(BRIEF | NORMAL | DEBUG)] commands can be executed anywhere in a command file, and you can change the logging mode more than once in a command file. When a command file is

completed, the LOG mode always returns the value it had before the command file began.

These commands can also be executed outside of a command file which will change the defaults for any command files executed during the session. In a session where no LOG commands have yet been executed, the defaults are:

LOG SCREEN = ON LEVEL = NORMAL

A LOG command with no parameters reports the current settings on the status line of the screen and in the journal file.

Effects of the LOG OFF Command

When the LOG mode is not turned off, ICED™ will insert comments in the journal file to indicate that control passed to a command file. Then each command in the command file that modified geometry or system settings will be logged into the journal file as well. The journal file will look similar to:

```
! @Q:\ICWIN\AUXIL\TEST.CMD
XSELECT OFF
SEL LAYER 100 ALL
DELETE
LEAVE
! Exit file Q:\ICWIN\AUXIL\TEST.CMD
```

See **Journaling and Data Recovery** for details on how to use the journal file to recover data.

Unless speed is a major concern, it is strongly recommended to leave the LOG mode on. The journal file is not only a history of what actions were taken with your design, but also a recovery mechanism if your system crashes during your session, or if you make a major mistake. It is best to have all commands executed logged into the journal file.

The only good reason to use LOG OFF is for speed during huge command files. Logging to the journal file does increase run time, but for a typical command file, the time increase is negligible.

We recommend that you restrict use of the LOG OFF command to large command files that do not create geometry that you may need to recover. The command files generated by the DRC (Design Rule Checker, available from IC Editors, Inc.) and NLE (Net List Extractor, part of the LVS tool available from IC Editors, Inc.) are good examples of command files where LOG OFF is appropriate. Since these command files generate only temporary geometry to indicate errors, you will never need to recover from an error in the command file.

Actions taken by command files that contain user interactions (e.g. a prompt for the user to supply a macro value or select a component) will not be automatically re-created by executing the journal file when the log mode is off. This is because the results of the user interactions will not be recorded in the journal file. **Do not use the LOG OFF command in these types of command files or you will prohibit automatic recovery of lost work with the journal file.**

If a command file that uses LOG OFF as the first command calls a nested command file, there is no way for the nested command file to turn the LOG mode back on.

If the LOG mode is on in command file, and a nested command file uses a LOG OFF command as the first statement in the file, the LOG mode will remain off until the nested command file is complete. The LOG mode in use by the first command file will then be back in effect.

Even when no LOG OFF command is included in a command file, you can turn logging off on the same line as the *@file_name* command. The LOG OFF command will then be executed as though it was the first command in the file.

Example: **@DRCOUT.CMD; LOG OFF**

The commands in DRCOUT.CMD will not be logged in the journal file. Only the call to the command file is logged to the journal file. In this case, the journal file will look similar to:

```
! @Q:\ICWIN\DRC\DRCOUT.CMD
! LOG OFF
@Q:\ICWIN\DRC\DRCOUT.CMD LOG=OFF
! Exit file Q:\ICWIN\DRC\DRCOUT.CMD
```

If this journal file is executed in a new ICED™ session to recover work, the command file will be called and re-executed. However, recovery of work done by that command file is not guaranteed. If the command file has been changed, the new version is executed. Or if a command file contains a SELECT command that requests the user to select a component, and the user does not select the same component he did the first time, the effect of the command file will be different the second time it is executed.

Showing Progress Messages During LOG OFF

The LOG OFF command also prevents commands from being echoed on the status line of the display screen. In a long command file, the session will appear "frozen" since no visible evidence that a command file is executing will appear on the display. The user may be confused by this, and even fail to realize when the command file is completed.

Adding *\$comment* commands to the command file cannot help this situation, since they will be prevented from being displayed on the screen. However, if you prefix the comment with double dollar signs ("\$\$") then the comment will show on the screen and in the log file even when the log mode is off. This is a good way to show progress indicators in your command file. See the *\$comment* command for more details.

Effects of the LOG SCREEN =(ON |OFF) Command

A LOG SCREEN=OFF command prevents all commands except for *\$comment* commands from being echoed on the status line of the display. This command does not affect the logging of commands to the journal file.

Turning off the logging of commands to the screen display with this command can speed up a command file considerably, and recovery of a crashed session is not compromised as it is in the case of the LOG OFF command.

In order to see the speed increase associated with a LOG SCREEN OFF command, the view mode should also be off. (I.E. No VIEW ON command should be in effect.) When the view mode is on, commands will still be echoed on the command line as they are executed, despite the execution of the LOG SCREEN OFF command.

If the view mode is off, and you have turned off the echo of commands to the status line with a LOG SCREEN OFF command, the only commands displayed on the screen are *\$comments*. You may want to add some *\$comment* or *\$\$comment* commands to display progress reports. Otherwise, the user will see no visible evidence that your command file is executing.

Effects of the LOG [LEVEL=(BRIEF | NORMAL | DEBUG)] Command

Several commands can optionally produce !comments in the journal file. (These comments are not shown on the screen.) These comments provide extra information that can be of interest when determining exactly what happened during a command, but they will not affect how the commands in the journal file are executed. Adding these comments to the journal file takes extra time during the execution of a command file.

Type of command	Example	LOG LEVEL		
		BRIEF	NORMAL	DEBUG
Commands that alter geometry or alter system settings	ADD, MOVE, GRID, LOG	Command is logged	Command is logged	Command is logged
Commands that alter view window	VIEW IN	Command is logged	Command is logged and !comment reports new window	Command is logged and !comment reports new window
Macro definition or assignment	#I = { %I + 1 }		!comment reports assignment	!comment reports assignment
Conditional execution	IF, WHILE			!comment reports if test was passed

Figure 66: Effects of log level on journal file for different commands

You can speed up a long command file by adding a LOG LEVEL=BRIEF command to your command file. This will prevent the generation of most !comments. This can decrease the size of the journal file considerably without affecting its ability to recover your work.

If you are having trouble debugging a command file, you can execute a LOG LEVEL=DEBUG command. This will add comments that help you follow exactly how the conditional statements were executed. The execution of conditional statement commands (IF, ELSE, ELSEIF, and WHILE) are not reflected in the journal file unless the LOG level is DEBUG.

Example:

```
LOCAL #FIRST_BOX $PROMPT="DIGITIZE FIRST BOX" BOX  
LOCAL #I = 1  
WHILE (%I < 4) {  
    ADD BOX OFFSET={%I * (0.5, 0.5)} AT %FIRST_BOX  
    #I = {%I + 1}  
}
```

See the
Command File
Programmer's
Reference
Manual for
details on these
command file
statements.

The lines above will add only 3 boxes, none of which will be at the location digitized by the user. Let us assume that this is not what the writer of the command file expected. If the lines above are executed when the log level is NORMAL (the default), the journal file will look like the following:

```
! LOCAL FIRST_BOX="(65.0,-55.0) (70.0,-50.0)"  
! LOCAL I="1"  
ADD BOX LAYER=WELL ID=163 OFFSET=(0.5, 0.5) AT (65.0,-55.0) (70.0,-50.0)  
! LOCAL I="2"  
ADD BOX LAYER=WELL ID=164 OFFSET=(1.0, 1.0) AT (65.0,-55.0) (70.0,-50.0)  
! LOCAL I="3"  
ADD BOX LAYER=WELL ID=165 OFFSET=(1.5, 1.5) AT (65.0,-55.0) (70.0,-50.0)  
! LOCAL I="4"
```

It is not immediately obvious why the loop executed only 3 times since no record of the execution of the WHILE command is reported in the log file.

If a LOG LEVEL=DEBUG command is added before the lines are executed, the log will add !comments each time the test in the WHILE command is evaluated. The journal file will then look like the following:

```

! LOCAL FIRST_BOX="(60.0,-45.0) (65.0,-40.0)"
! LOCAL I="1"
! WHILE(1){ -- Begin block -- Line 4
ADD BOX LAYER=WELL ID=160 OFFSET=(0.5, 0.5) AT (60.0,-45.0) (65.0,-40.0)
! LOCAL I="2"
! } -- End WHILE block -- Line 7
! WHILE(1){ -- Begin block -- Line 4
ADD BOX LAYER=WELL ID=161 OFFSET=(1.0, 1.0) AT (60.0,-45.0) (65.0,-40.0)
! LOCAL I="3"
! } -- End WHILE block -- Line 7
! WHILE(1){ -- Begin block -- Line 4
ADD BOX LAYER=WELL ID=162 OFFSET=(1.5, 1.5) AT (60.0,-45.0) (65.0,-40.0)
! LOCAL I="4"
! } -- End WHILE block -- Line 7
! WHILE(0){ -- Skip to end of block -- Line 4
! } -- End of block -- Line 7

```

Now the writer can see exactly what the value of the macro I was during each execution of the loop. The line numbers mentioned in the !comments about the WHILE block refer to the line number of the WHILE statement in the command file.

MENU

Load a menu file.

MENU ["*menu_name*" | * [:"*submenu_name*"]]

See the **MkMenu** utility for details on how to create a menu file.

The MENU command is used to load a MENU file containing a customized command menu. The file *menu_name*.MEN must already exist in the AUXIL subdirectory.

The quotes around *menu_name* are optional. Use them if your menu name contains blanks or other characters that might confuse the command parser.

See page 44 for details on the search path for auxiliary files.

If you use the MENU command, you should be aware of how ICED™ loads menus. If a menu is specified with the MENU option on the editor command line, that menu is loaded first. If this is a new cell, the startup command file is then executed. Any menu specified by a MENU command in a startup command file will override a menu already loaded.

The menu file reference in use when a cell file is stored, is saved in the environment database of the cell. The menu reference stored in an existing cell will be used by default in future edit sessions. However, when you edit an existing cell and a menu is specified on the layout editor command line, it will override the menu stored with the cell file.

Example:

MENU DRC

The DRC menu is a special menu used for executing the DRC program (a separate product available from IC Editors used to perform advanced layer processing and design rule verification) from within the layout editor. This command will replace the existing menu with the menu definitions stored in DRC.MEN. This menu has the same first three top-level menus as M1.MEN. A fourth top-level menu is added with special options for DRC processing.

You can use "*" instead of the menu name to refer to the current menu. This is used primarily in conjunction with the optional `:"submenu_name"` parameter. When a submenu is specified in the MENU command, a submenu specified in the menu file is temporarily loaded in front of the other top-level menus in the file. A submenu with this name must already be defined in the menu file.

Example: **MENU *:DRC_STEP**

This command executed after the example above will temporarily place a new top-level menu in front of the other four. The DRC_STEP submenu has several special options for stepping through the list of design errors found by the DRC.

Example: **MENU ***

If this command is executed after a MENU command specifying a submenu, it will remove the previously loaded submenu from the list of displayed top-level menus. (In the rare case when no menu is currently loaded, MENU * loads the default M1 menu.)

The submenu setting is not preserved in the environment database of the cell file. If the cell is saved when a submenu is loaded, when the cell is reopened, the submenu will no longer be loaded in front of the other top-level menus.

Example: **MENU**

Executing the MENU command with no parameters will display the current menu settings.

MERGE

Merge wires or lines. Merge or subtract polygons.

MERGE (WIRES | LINES | POLYGONS)

The MERGE command will join 2 components into a single component. The only types of components you can join are wires, lines, polygons, and boxes (Boxes are a special case. They can be considered either polygons or wires by the MERGE command.) The choice of which type of components will be joined by the MERGE command is not optional. One of the WIRES, LINES, or POLYGONS keywords must be used in the command.

MERGE WIRES

The DRC can be used to merge wires with other shapes without restrictions. Refer to the Internal DRC tutorial in the Classroom Tutorials Manual.

The MERGE WIRES command will join two selected components into a new wire component. Both original components must be either wires or boxes. The two components must have the same width and they must be on the same layer.

The components can be selected prior to executing the MERGE WIRES command. If no components are selected when you execute the MERGE WIRES command, an embedded SELECT END IN command is generated to allow you to select the ends of the wires to join.

Only one end of each wire can be selected. If both ends are selected, ICED™ cannot determine which end to join and will issue an error message. Whether or not the sides of wires are selected will not affect the MERGE WIRES command.

If you are merging a box as a wire component, you must select the side of the box prior to issuing the MERGE WIRES command. (This is because a box component has no 'end', so a SELECT END IN command will never select a side of a box.) The selected side must be the same width as the wire. It is also permissible to have three sides of the box selected. In this case, the middle of the three sides will be treated as a wire end.

Example: **SELECT END IN
MERGE WIRES**

In this example the wire ends to merge are selected in a separate SELECT command before executing the MERGE command. Since the SELECT command uses the END keyword, only the wire ends are selected. The IN keyword allows you to use the mouse to pick the corners of the box shown in Figure 67. This use of the SELECT command will select all wire ends within the boundaries of the box. The result of the SELECT command is shown in Figure 68.

The MERGE command will calculate the projected intersection of the wires and will create the new wire component shown in Figure 69. The original wire components are deleted.

ICED™ must be able to project an intersection to perform the merge. (For example, the wire segments selected for the merge should not be parallel but offset from each other.) Portions of the original wire segments that extend past the intersection are trimmed back to the intersection as shown in Figure 69.

Be aware that the merge command will not prevent you from creating self-intersecting components.

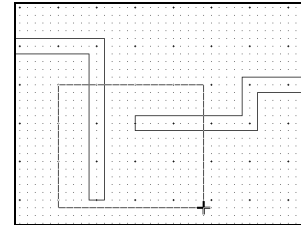


Figure 67: Using the select box to select wire ends.

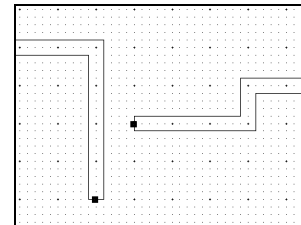


Figure 68: Select marks show the selected ends of the wires.

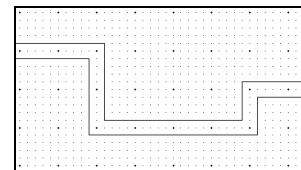


Figure 69: Result of the MERGE wires command.

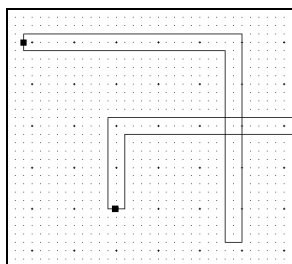


Figure 70: Wire ends selected for merge.

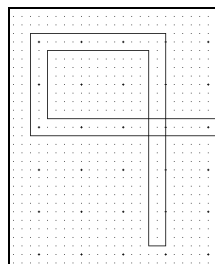


Figure 71: Self-intersecting wire created.

MERGE LINES

MERGE LINES works in a very similar manner to MERGE WIRES. The ends of exactly two line components must be selected. (As long as only two ends are selected, selected line segments in the middle of the lines will not affect the merge.) If the line ends are not selected prior to executing the MERGE command, ICED™ will generate an embedded SELECT END IN command to allow you to draw a box around the selected ends of the lines.

The line segments connected to the ends of the lines must not be parallel, or ICED™ cannot determine an intersection.

MERGE POLYGONS

The DRC can be used to merge and/or etch shapes without a common side. Refer to the Internal DRC tutorial in the Classroom Tutorials Manual.

Polygons are handled differently by the MERGE command than lines and wires. ICED™ can use one polygon to remove area from another, or it can add two polygons together. The two polygons **must** share at least one common side. If the polygons do not overlap, the new polygon will be the sum of both without a seam on the shared side. If the polygons do overlap, ICED™ will remove the area of intersection. Some polygon intersections may result in invalid components and ICED™ may issue an error message instead of performing the merge. However ICED™ does not recognize all possible invalid components.

To use the MERGE POLYGONS command, exactly two components on the same layer must be selected. The two components must be either polygons or boxes. At least one side of each polygon must be selected such that those two sides overlap. (See examples below.) Other sides of the two polygons may be selected as well.

The components may be fully or partially selected before the MERGE command executes. If no components are selected when the MERGE POLYGONS command is executed, ICED™ will issue an embedded SELECT SIDE NEAR command. This form of the SELECT command will select component sides using a box whose center is the cursor on the screen. This box is called the near box. Click the left mouse button once when the cursor is over the common side of the two polygons. This should result in the selection of sides from both polygons with a single click.

To set the size of the near box, see the **NEAR** command.

Example: **MERGE POLY**

If no components are selected when this command executes, the near cursor (a cursor symbol with a small box attached) will appear on the screen. Click the left mouse button when the cursor is over the common side of the two polygons. If the polygons do not overlap as shown in Figure 72, the new polygon will be the sum of the two original polygons. See Figure 73. If the two polygons do overlap as in Figure 74, then the new polygon will be the area of the larger polygon with the smaller polygon subtracted from it. See Figure 75. The two original polygons will be deleted at the end of the command.

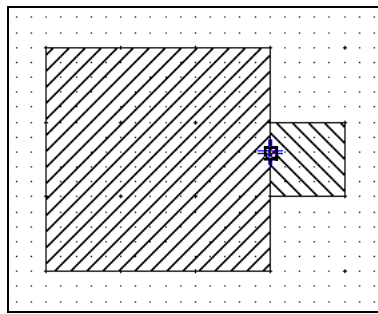


Figure 72: Using the cursor to select polygons for merge.

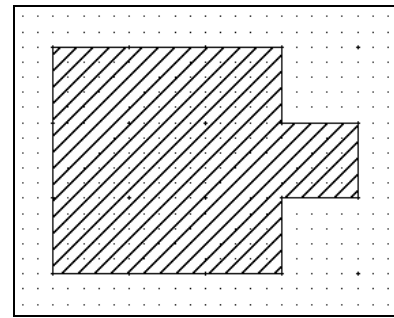


Figure 73: The new polygon is the sum of the two polygons.

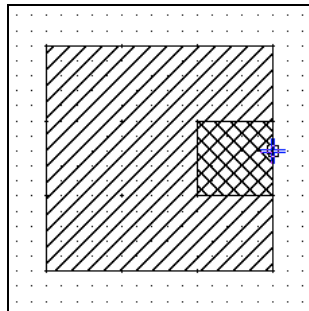


Figure 74: Merging overlapping polygons.

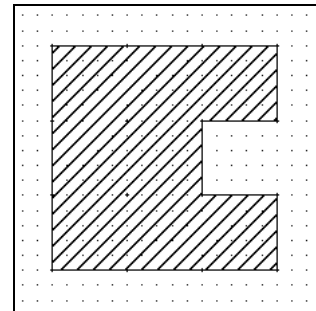


Figure 75: The new polygon is the difference of the two polygons.

The DRC can be used to etch holes, or shapes without a common side. Refer to the Internal DRC tutorial in the Classroom Tutorials Manual.

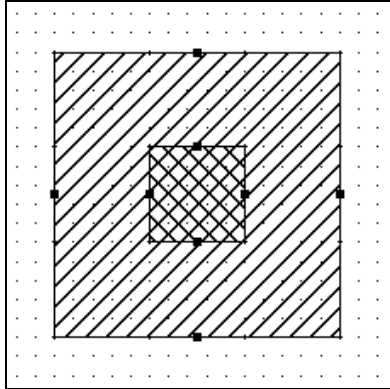


Figure 76: This merge will fail because there is no common side.

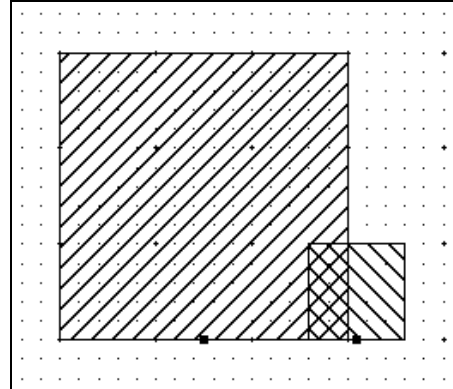


Figure 77: This merge will fail because the result would be an invalid polygon.

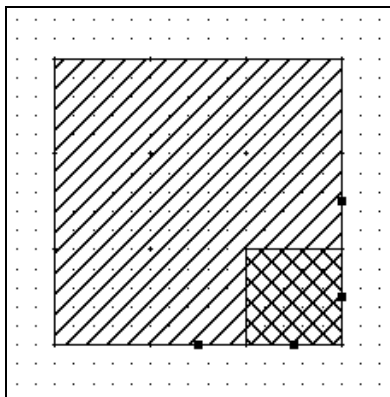


Figure 78: This merge will succeed even though two sides overlap.

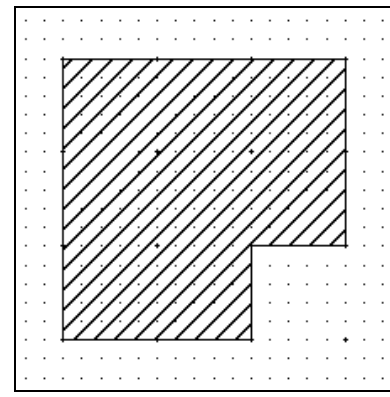


Figure 79: Result of the merge in Figure 78.

MIRROR

Move components by mirroring about an axis.

MIRROR (X [=x_coord] | Y [=y_coord])

See **COPY MIRROR** to copy rather than move components.

The MIRROR command relocates all fully selected components by mirroring them about an axis. The axis of the mirror is either a vertical line: MIRROR X, or a horizontal line: MIRROR Y. One of the keywords, X or Y, is required.

Example: **MIRROR Y=10.0**

This command relocates all fully selected components by mirroring them about the line Y=10.0. Note that y=10 is a horizontal line. Thus, MIRROR Y results in mirroring the components about a horizontal axis.

If you do not specify the mirror axis coordinate as part of the MIRROR command, or if you select the command from the menus, you must enter the mirror position with the mouse. The following command expects mouse input:

Example: **MIRROR X**

In this example, the polygon in Figure 80 was selected prior to executing the MIRROR command. Since the x_coord parameter was not provided in the command, the vertical mirror line appeared on the screen to define the axis. Once the mouse button is pressed to select the x-coordinate of the mirror axis, the component is relocated. The result of this command is shown in Figure 81.

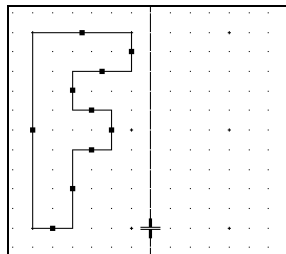


Figure 80: Using cursor to define axis with MIRROR X.

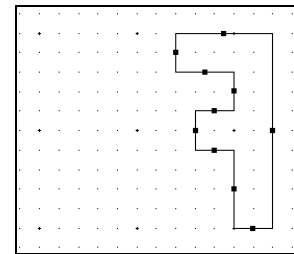


Figure 81: The result of the MIRROR X command shown in Figure 80.

See the **NEAR** command to set the size of the NEAR box.

If you do not have any components selected prior to executing a MIRROR command, an embedded SELECT NEAR command will be generated by ICED™. Click the left mouse button when the cursor is over an edge of the desired component. The SELECT NEAR command will select any unprotected components within the near box. If you wish to mirror several components, or only one component in a crowded area, you can select the component(s) with the SELECT command prior to executing the MIRROR command.

MOVE

Move existing components or their vertices.

MOVE [SIDE | VERTEX] [X | Y] [[BY] *disp*]

See the **MIRROR** and **ROTATE** commands for other methods of moving entire components.

Example:

The MOVE command is used to perform two different operations. Fully selected components will be relocated to a new location. Partially selected components will have their selected sides shifted.

You can use SELECT commands to select whole components or just component side(s) prior to executing the MOVE command.

SELECT SIDE MOVE

If the SELECT SIDE command is used to partially select a wire as shown in Figure 82, the MOVE command can then be used to shift the selected side. Once the MOVE command is executed, the cursor is used to indicate two positions to define the displacement. Only the horizontal displacement is used since the wire segment is vertical.

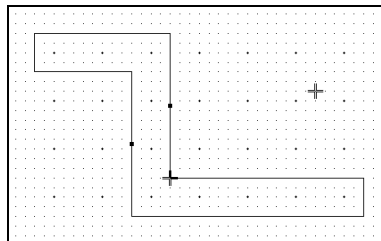


Figure 82: Using cursor to define displacement for partially selected wire.

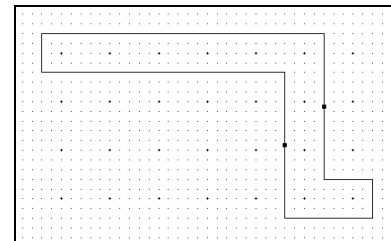


Figure 83: Result of MOVE command in Figure 82.

[SIDE | VERTEX]

If no components are selected prior to executing the MOVE command, and you do not use the SIDE or VERTEX keywords, an embedded SELECT NEAR command allows you to select only entire components. However, either the SIDE or VERTEX keyword in a MOVE command will cause ICED™ to generate an embedded SELECT SIDE NEAR command instead.

If you select two sides by pressing the left mouse button while the cursor is at a vertex, the vertex in between the sides will be moved. If you select all sides of a component, the entire component will be moved.

MOVE SIDE allows you to shift sides of a component in a direction parallel to the original sides. MOVE VERTEX allows a vertex to move so that the sides are at an angle different than the original sides. (See figures below.)

Example:

MOVE SIDE
MOVE VERTEX

Both of these commands generate an embedded SELECT SIDE NEAR. To select a corner of a box, you click the left mouse button while the cursor is over the corner. To define the displacement of the move, click the left button when the cursor is 5 units up and to the right, as shown in Figure 84. The MOVE SIDE command will create the large box shown in Figure 85. MOVE VERTEX will create the polygon shown in Figure 86.

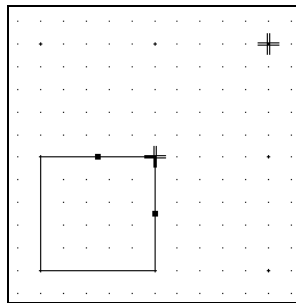


Figure 84: Using the cursor to define displacement for a MOVE command.

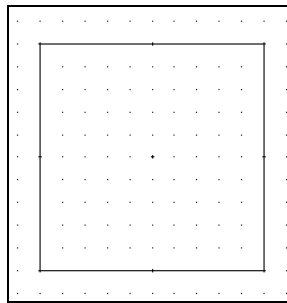


Figure 85: Result if MOVE SIDE was used.

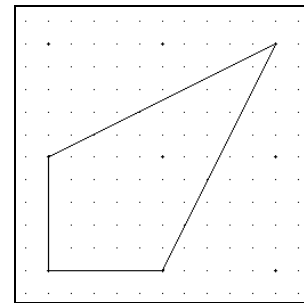


Figure 86: Result if MOVE VERTEX was used.

[X | Y]

Using the X or Y keywords tells ICED™ to restrict the move to one axis. If you specify a single axis with the X or Y keywords, then displacement along the other axis is ignored. This is true whether the displacement is entered with the mouse or with the *disp* parameter. (An example of using these keywords follows later in this chapter.)

[[BY] *disp*]

If you type the MOVE command from the keyboard (or in a command file) you can include the displacement, *disp*, as part of the command. If the X or Y keywords are not used, then *disp* is a pair of real numbers. The first number is displacement in the X direction. The second number is displacement in the Y direction.

Example: **MOVE BY (2.5, 10.0)**
 MOVE (2.5, 10.0)
 MOVE 2.5 10.0

These three examples are equivalent. Each command moves selected components by 2.5 units in the X direction and 10 units in the Y direction. Note that the BY keyword is optional. It is provided to make the command more readable.

If the X or Y keyword is used to restrict movement to one axis, then *disp* is only a single real number, the displacement along the selected axis.

Example: **MOVE Y 10.0**

This example moves the selected components, sides, or vertices by 10 user units in the Y direction.

If you do not specify the displacement as part of the MOVE command or if you select the command from the menus, you will have to enter the *disp* parameter with the mouse. If you use the mouse, the displacement is entered by selecting two points. Any two points can be used, but the result will be easier to visualize if you put the first point on a selected component and the second point where you want it to end up. The differences between the X values of the two points and the Y values of the two points become the X and Y displacements. If the X or Y keywords are used, displacement in the other direction is ignored.

Example: **MOVE X**

The fully selected component in Figure 87 (the 'F') was selected prior to executing the MOVE X command. Once MOVE X is executed, the cursor is used to define the point on the selected component as the reference point for the displacement. The cursor is then used to define the point on a second component (the 'T'). The displacement in the Y direction is ignored which has the effect of aligning the two components vertically.

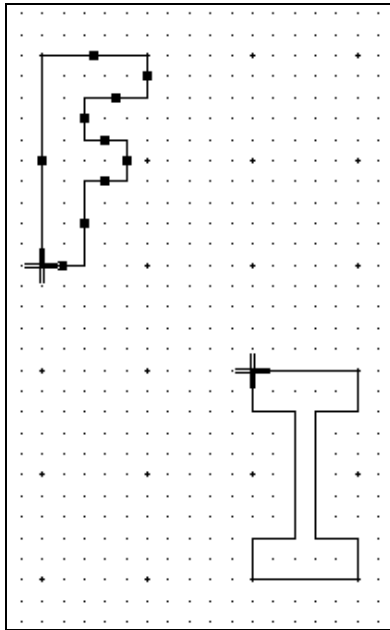


Figure 87: Using cursor to align components with MOVE X.

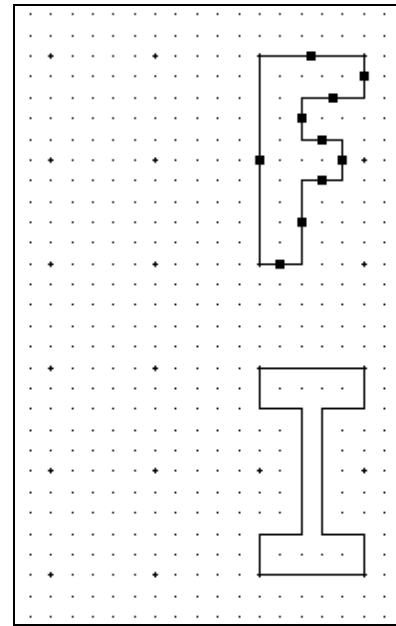


Figure 88: Result of MOVE X in Figure 87.

When you use the MOVE command without providing the displacement, and no components are selected prior to executing the command, the point used to select the component is used as the first displacement reference position. In the example above, if no component was selected prior to executing the command, the first mouse click would have selected the polygon and defined the first reference point. The second mouse click would complete the command.

Situations Where the MOVE Command Will Fail

The MOVE command can fail in certain situations. When this happens, an audible beep will sound and a message similar to the one below will be displayed:

*****Move failed for 2 component(s)*****

No components will be moved if any component causes an error. You can determine exactly which components caused the failure with a SELECT FAIL command.

Several types of MOVE failures are listed below:

Box inversions - You cannot shift one side of a box beyond a far side of the same box. If you define the displacement of the MOVE SIDE shown in Figure 89 by clicking at point 1 then at point 2, the MOVE command will fail.

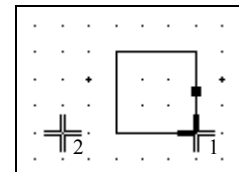


Figure 89: A box inversion will fail.

See an example of off grid coordinates in the MOVE SIDE tutorial in the **Classroom Tutorials Manual**.

See the **NDIV** command line parameter for an explanation of the range for valid coordinates.

Calculated vertices off grid - If you perform a MOVE SIDE command on a polygon with sides at an angle other than horizontal or vertical, a MOVE SIDE will rarely be able to create the calculated vertex on the resolution grid, so the command will fail.

New coordinates beyond valid range - Valid coordinates must lie within the maximum size of the cell. If a MOVE command results in a component vertex outside the boundary of the drawing, the command will fail.

Line segments with 0 length - If you shift line segments such that 2 or more consecutive segments have 0 length, the MOVE may fail.

NEAR

Set the half width of the near box.

NEAR [UNITS=*units*] [DOTS=*dots*]

The NEAR command is used to control the half width of the near box used in the SELECT NEAR command. The SELECT NEAR command is the most common form of the embedded SELECT command.

The SELECT NEAR command selects all objects that are in or whose edges cross a small square. This square is called the 'near box'. The half width of the near box is specified as the larger of two values: an absolute size in user units and an apparent size in dots (pixels on the screen). This definition prevents the near box from becoming uselessly small when working with large view windows.

The *units* parameter must be a real number in the range 0:10. The *dots* parameter must be an integer in the range 0:20.

Example: **NEAR UNITS=.5 DOTS=4**

This command specifies that the near box is 1.0 user units wide or 8 pixels wide, whichever is larger.

Both the UNITS and DOTS keywords are optional. If either is omitted, its value remains unchanged.

Example: **NEAR**

This form of the NEAR command (with no parameters) can be used to display the current values of UNITS and DOTS on the screen.

OUTLINE

Set depth for which cell and array outlines are drawn.

OUTLINE [[DEPTH=] *depth*]

Refer to the **BLANK** command for details on blanking (hiding) cells.

ICED™ can draw dotted white bounding boxes around any unblanked cells and arrays. These boxes can help you see which components are contained in cells. However, if you have cells nested within cells, nested within other cells, these outlines can get confusing. The OUTLINE command is used to set the depth for which ICED™ draws these bounding boxes. (The bounding boxes can usually be considered to be on layer 0. This layer will never be included in CIF, Stream, or DRC export. However, layer 0 can be plotted.)

[[DEPTH=] *depth*]

See the **VIEW LIMIT** and **DISPLAY** commands to control the depth for which components in cells will be drawn.

Cells and arrays that are placed in the root cell are said to have depth 0. Cells and arrays contained in depth 0 cells have depth 1 and cells and arrays contained in depth 'n' cells are said have depth 'n+1'. Similarly, the cells that form an array of depth 'n' have depth 'n+1'.

ICED™ will draw bounding boxes for unblanked cells and arrays that have a depth **less than** *depth*. *depth* must be an integer in the range 0:100.

Example:

OUTLINE DEPTH=0

ICED™ will not draw bounding boxes for any unselected cells or arrays. Selected cells and arrays will have bounding boxes displayed.

Example:

OUTLINE 1

If *depth*=1, ICED™ will draw bounding boxes for unblanked cells and arrays placed in the drawing you are editing. However, it will not draw bounding boxes for cells contained within these cells or arrays. Note that the DEPTH keyword is not required to specify *depth*.

Example: **OUTLINE DEPTH=100**

If *depth*=100, ICED™ will draw bounding boxes for all unblanked cells and arrays.

Example: **OUTLINE**

This form of the OUTLINE command (with no parameters) can be used to display the current value of the outline depth.

If you want cell outlines to display in your plots, assign a non-negative pen number to layer 0.

P_EDIT

Edit another cell in place.

P_EDIT [NOW] [LOCAL_COPY=(TRUE | FALSE)] ...
 ... [VIEW_ONLY=(TRUE | FALSE)] (SELECT | NEAR [position])

See the **EDIT** command for a chart comparing the 3 different edit commands.

The P_EDIT (EDIT in **P**lace) command allows you to suspend work on the cell you are currently editing and edit a nested cell in place. (We will call these cells the parent cell and the child cell. We will call the cell you were editing when you first launched ICED™ the root cell.) While you are editing the child cell, the parent cell will still display on the screen. This will allow you to modify the child while viewing the parent cell for reference. All copies of the child cell in the parent cell will be updated on the screen as you modify it.

The **FILL MIXED** command can be used to turn fill patterns on only in the cell being edited.

During a P_EDIT session, you will use the coordinate system of the parent cell instead of the coordinate system of the child cell. All coordinates will be reported relative to the origin of the parent cell rather than the origin of the child cell.

When you use the P_EDIT command to edit a cell, ICED™ loads only the cell geometry database. The child cell's environment database, as well as its coordinate system, is ignored. Thus, if layer 1 is red in the parent cell and blue in the child cell, it will appear red while you use P_EDIT to edit the child cell.

The **T_EDIT** command is similar to P_EDIT, but the parent cell is not displayed on the screen.

You cannot suspend a P_EDIT session. That is, you will not be able to use the EDIT, T_EDIT, or P_EDIT commands to edit a different cell until you return from editing the child cell.

When you are done editing the child cell, you can use the QUIT, EXIT, or LEAVE commands to return to the parent cell. If you QUIT the child cell, the changes you made will be lost. If you use EXIT or LEAVE, ICED™ will save your changes in RAM. (LEAVE will save the cell only if the geometry has been modified.) These changes will be saved to disk when the editor terminates even if you eventually QUIT the root cell.

See page 45 for more information on protected cell libraries.

If the child cell is in a different cell library than the root cell, you may be restricted from saving changes made to it. The P_EDIT command will prompt you with a warning message if the cell is in a protected library.

[LOCAL_COPY=(TRUE | FALSE)] [VIEW_ONLY=(TRUE | FALSE)]

These parameters are used mainly in command files. You can skip this information if you are not using P_EDIT in a command file.

If you prefer to avoid having ICED™ display a prompt if the cell you wish to edit is in a protected library, you can use the LOCAL_COPY or VIEW_ONLY keywords. When these keywords are not used in a P_EDIT command, and the cell is in a protected library, you will see a prompt similar to:

Cell_name already exists in Q:\path_name

Enter option (V=>VIEW only; L=>edit LOCAL copy; C=>CANCEL):

The LOCAL_COPY=TRUE parameter will automatically create a local copy of a cell from a copy edit or direct edit library. This has the same effect as replying with an <L> to the above prompt. When you exit the editor, the cell file will be saved in the same directory as the root cell.

Using the LOCAL_COPY=FALSE parameter has no effect on the P_EDIT command. The warning prompt will be displayed if no VIEW_ONLY=TRUE parameter is present.

If the VIEW_ONLY=TRUE parameter is used, you will automatically be placed in view-only mode. You will not be able to use the EXIT or LEAVE commands to save changes to the cell. Using this parameter has the same effect as replying with a <V> to the above prompt. It will allow you to view cells in any library (including a view-only or copy-edit library) without a warning prompt.

Using the VIEW_ONLY=FALSE parameter has no effect on the P_EDIT command. The warning prompt will be displayed if no LOCAL_COPY=TRUE parameter is present, or if the cell is in a view-only library.

When the LOCAL_COPY or VIEW_ONLY keywords are used in a P_EDIT command, they should precede the SELECT or NEAR keyword.

SELECT

The name of the cell you are editing is shown in the upper left corner of the view window. All open cells are listed to the left of the name of the current cell.

One of the keywords SELECT or NEAR must be used with the P_EDIT command.

The P_EDIT SELECT command allows you to edit a selected cell. Only cells can be selected when you issue this command. If there is more than one cell selected when the P_EDIT SELECT command is executed, ICED™ will assume you want to edit the selected cell with the smallest area. If no cells are selected when the P_EDIT SELECT command executes, an embedded SELECT CELL * AT command is generated to allow you to select the cell.

NEAR [*position*]

See **EDIT NEAR** for more details and an example.

The P_EDIT NEAR command is the most common form of the P_EDIT command. The NEAR keyword allows you to traverse several levels of cell hierarchy at a time. If *position* is the coordinate pair that identifies a point on the edge of a visible component in a nested cell, P_EDIT NEAR *position* will allow you to edit the deeply nested cell directly. You usually use the mouse to digitize *position* instead of supplying it with the command.

The NOW Keyword and the ENTER.SUBCELL Macro

See the EDIT command for more details on the ENTER SUBCELL macro and the NOW keyword.

The following new feature is useful primarily if you use specialized command files and/or macros. This feature is described in more detail in the description of the EDIT command.

If an ENTER.SUBCELL macro is defined when a P_EDIT command is executed, the string stored in the macro is executed as a command string. If the macro does not exist, the P_EDIT command continues without any errors. Add the NOW keyword to the P_EDIT command to avoid executing the macro.

Example: **P_EDIT NEAR**

When the command above is used to edit a cell, and the macro ENTER.SUBCELL exists, then the command string stored in the macro is executed as soon as the cell is opened. If the command string includes a *@cmd_file* command, all of the commands in the file are executed.

Example: **P_EDIT NOW NEAR**

When the NOW keyword is added to the P_EDIT command, the ENTER.SUBCELL macro is ignored and the P_EDIT proceeds without executing the command file.

PACK

Compress buffer space; report on memory usage.

PACK

ICED™ stores data in 8K byte memory blocks called pages. Under certain unusual circumstances, it is possible for an ICED™ database to use a large number of mostly empty pages. If this happens, executing the PACK command will improve ICED™'s performance.

The PACK command will also tell you how much memory the database requires.

The data produced by the PACK command is reported at the bottom of the ICED™ screen and copied to the journal file. It will look similar to:

PACK ! Database requires 7.2M-bytes

You may be able to provide ICED™ with more memory using advanced command line parameters when you launch ICED™.

See the
WINDOWS,
ICED, and
RAM command
line options to
learn more
about memory
management.

PATTERN

Load a stipple pattern file.

PATTERN *pattern_name*

ICED™ can display geometries using solid fill, no fill (outlines only), or user definable stipple patterns. The definitions of the stipple patterns are stored in a stipple pattern file (.STI file). The MkSti utility is used to create the .STI file, the PATTERN command is used to load it into memory, and the LAYER command is used to assign specific patterns to individual layers. The FILL command is used to enable or disable pattern fill.

When you load a new .STI file, any existing pattern definitions are lost.

See page 44 for details on the search path for auxiliary files.

The file *pattern_name*.STI must already exist in the Q:\ICWIN\AUXIL directory. ICED™ comes with two predefined .STI files, SAMPLE.STI and NONE.STI. SAMPLE.STI has a number of useful and illustrative pattern definitions. NONE.STI does not contain any pattern definitions. It is used primarily to plot layouts in outline mode without patterns.

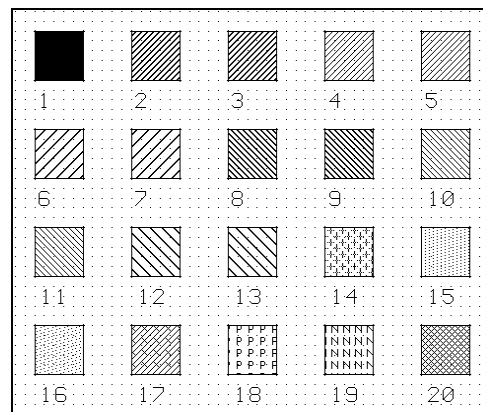


Figure 90: The first twenty stipple patterns in the SAMPLE.STI file.

Example: **PATTERN SAMPLE**
 LAYER M1 PATTERN=2 COLOR=BLUE
 LAYER M2 PATTERN=3 COLOR=CYAN

The PATTERN command will load the stipple patterns shown in Figure 90. (Only some of the available patterns are shown.) The first LAYER command assigns the pattern labeled with a 2 to the M1 layer. All components on that layer will now be displayed with that pattern filling the outlines if the FILL command has enabled pattern fill. The next LAYER command assigns fill pattern 3 to the M2 layer. Note that fill patterns 2 and 3 are very similar, but slightly offset from each other. If a component on the M1 layer overlaps a component on the M2 layer, the fill patterns (and colors) for each component will be visible since the fill patterns will not be exactly on top of each other.

The **FILL** command turns the screen display of fill patterns on or off.

If components on layers with patterns assigned to them are still drawn with outlines only, toggle the pattern fill parameter with the following shorthand for the FILL command:

Example: **F**

The patterns loaded by this command affect screen display only. The same *pattern_name*.STI file can be used by the PLOT command. Assign the patterns for plotting to specific layers with the PEN parameter of the LAYER command.

PAUSE

Create a pause in a command file.

PAUSE [*seconds*] ["*msg_string*"]

See the
@file_name
command for
details on using
command files.

The PAUSE command is used to insert wait states in command files. It is usually used to give a command file user a chance to see a message. (The command will have no effect if typed in during an ICED™ session except that it will be logged into the journal file.)

If used, the *seconds* parameter determines how long ICED™ will be frozen. The *seconds* parameter must be a real number in the range 0:60. When you omit *seconds*, or specify '0', then the command file is paused until the user hits a key or mouse button. A prompt indicating this replaces the command prompt near the bottom of the screen.

Example:

**\$ This message is left on history line during pause
PAUSE**

These lines will leave the *\$comment* displayed on the history line until the user presses a key or mouse button. The menu is not displayed. Once the user has interrupted the pause, the execution of the command file continues.

When you supply a *msg_string*, it is displayed on the command line during the wait. Use quotes around *msg_string* if it contains syntax that might confuse the parser like other quote characters.

If *seconds* is non-zero, the command file will be paused for that length of time before the command file automatically continues. The user can interrupt the wait by clicking the mouse or typing any key on the keyboard. However, no message informing the user of this is displayed unless you supply an appropriate *msg_string* in the command.

Example: **\$ This message is left on history line during pause**
 PAUSE 10 Press key or mouse button to continue immediately.
 \$ This message is displayed on history line after pause is done

These lines will display the first two messages for 10 seconds. A countdown in yellow replaces the command prompt. When the countdown gets to 0, the execution of the command file continues.

The command parser performs macro replacement in the *msg_string*. You should surround the *msg_string* with quotes if it uses syntax that may cause parsing conflicts.

Example: **PAUSE "%MYMAC errors found"**

See the
Command File
Programmers
Reference
Manual for
details on
macros.

Before this command is executed, the value of the MYMAC macro will replace the %MYMAC reference in the message string. Since the example above omits the *seconds* parameter, if MYMAC contained a number, and the quotes were not present, the parser would interpret the number as the *seconds* parameter of the PAUSE command.

PLOT

Create an intermediate file for the MkPlot utility.

PLOT PLOTTER=*pdf_name* ...
... PATTERN=*pattern_name* ...
... SIZE=*size* ...
... [DOTS=(1 | 4) | DITHER]
... [LAYERS=*layer_list*] ...
... [NX=*nx*] [NY=*ny*] ...
... [STRIP= (YES | NO)]
... [R=(0 | 1)] ...
... [FILTER=*filter_size*] ...
... [DEPTH=*depth*] [OUTLINE_DEPTH=*out_depth*] ...
... [CELL_LABELS=(ON | OFF)]
... [ARRAY_MODE=(BLANK | FULL | SIDES)] ...
... [FILE=*file_name*] ...
... [TEXT_ORIENTATIONS=(2 | 4 | 8)] ...
... [TEXT_HIGHLIGHTS=(ON | OFF)] ...
... [METRIC | ENGLISH] ...
... (ALL [=*scale*] | SCALE=*scale* | PRESCALE) ...
... [[AT] *pos*]

See the distribution file Q:\ICWIN-\AUXIL\PLOT-NOW.CMD for a sample command file that executes both a PLOT command and the MkPlot utility.

The PLOT command causes ICED™ to create an intermediate file with a .VEC extension for use by the MkPlot utility. You must run the MkPlot utility (using the .VEC file as input) to actually send the data to the printing device.

The easiest way to execute the PLOT command is to use the ICED™ menu option 1:FILE → (PLOT)now. (Select the FILE option from the first top-level menu, then select "now" under the PLOT heading.) This executes the PLOTNOW.CMD command file. This command file also executes the MkPlot utility automatically so that you will not need to execute it as a separate step.

The 1:FILE → PLOT menu option executes only the PLOT command rather than the PLOTNOW command file that includes both the PLOT command and

the MkPlot utility. Using either menu option will automatically provide you with the correct options given your choice of plotting devices. The PLOT command can also be typed in or executed from a command file.

One easy way to get the correct syntax for a PLOT command is to generate the PLOT command with the menu option 1:FILE → PLOT, then use the <Page Up> key to have ICED™ echo back the command syntax it generated from your menu choices. You can also look in the journal file for exact syntax of the PLOT command generated by your menu choices.

See page 99 to learn about the journal file.

Once you have selected PLOT from the menus, your first option is the choice of a plotting device. (Since the list of plotters and printers supported by ICED™ is constantly growing, check the Q:\ICWIN²³\DOC\PLOTTERS.TXT file for which option to choose if you are not sure which one matches your printer or plotter.) The choice you make from the list of devices will create a PLOTTER=*pdf_name* option (we cover this later) in the generated PLOT command.

If your plot command fails with the message "No data on output layers" check that your layers are not all defined with the default value of NO_PEN. See page 271 and the LAYER command.

You will then be prompted to choose a pattern file for fill patterns. (We cover this subject later on with the PATTERN=*pattern_name* parameter.)

The next prompt (labeled SUPER-PIXEL Mode) determines the DOTS option that determines the color mode for plotting. This affects the number of colors plotted and the print resolution. (See the DOTS parameter on page 272 for more information.)

The following SIZE prompt is for the choice of paper sizes. ICED™ will automatically list the correct options for your plotting device.

At this point, the rest of the PLOT keywords are optional. The ICED™ menu will give you the following choices for setting the rest of the PLOT options: **none**, **some**, **most** or **all**. If you choose **none**, ICED™ will not present further menu options, and will use the ALL=* option (see page 280) to plot all of the data at a scale that will fit on your selected size paper. Figure 91 shows which PLOT options you can set using the **some**, **most** or **all** choices.

²³ Remember that Q:\ICWIN represents the drive and path where you have installed ICED™.

Parameter keyword	Purpose	Plot menu options that allow you to define parameter
NX and NY	used to plot the data over several pages	some, most or all
ENGLISH METRIC	determines the units of the scale factor	some, most or all
ALL SCALE PRESCALE	determines the scale and area to be plotted	some, most or all
LAYERS	selects which layers will be plotted	most or all
R=(0 1)	rotates plot	most or all
FILTER	filters components based on size	all
DEPTH and OUTLINE_DEPTH	limits depth to which nested cells and outlines are plotted	all
ARRAY_MODE	determines how cells in arrays are plotted	all
FILE	overrides the default .VEC output file name	all
TEXT_ORIENTATIONS	controls the orientation of text components	all
TEXT_HIGHLIGHTS ²⁴	Adds halo of white behind text	some, most or all
CELL_LABELS	Adds text labels in place of cells not drawn due to DEPTH restrictions	all

Figure 91: PLOT options available from the menu.

The keywords are described in detail below. These descriptions describe the syntax required to type in the PLOT command or use it from a command file.

²⁴ This is indicated by the title TEXT HALOS on some menus.

PLOTTER=pdf_name

See the file PLOTTERS.TXT in the Q:\ICWIN\DOC directory of your ICED™ installation for recommendations on *pdf_name* selection.

The PLOTTER parameter is required by the PLOT command. Each plotting device supported by ICED™ has at least one plotter description file with a .PDF extension associated with it. These files are stored in the Q:\ICWIN\AUXIL²⁵ directory. ICED™ will look for a plotter definition file with the file name *pdf_name*.PDF. If it does not exist, ICED™ will report an error. The directory path name and .PDF extension are NOT included in *pdf_name*.

PATTERN=pattern_name

The **LAYER** command's PEN keyword assigns fill pattern numbers to layers.

The required PATTERN keyword is used to select the auxiliary file of pattern definitions used to draw layers. You can use PATTERN=NONE if you do not want fill patterns used in your plot.

Refer to the **MkSti** utility description to create your own fill patterns.

The MkPlot utility, which handles printing the file created by the PLOT command, can use user-definable fill patterns. The pattern definitions are stored in files with .STI extensions in Q:\ICWIN\AUXIL²⁵ directory. These pattern files are created by the MkSti utility supplied with ICED™.

ICED™ will search for a pattern file with the file name *pattern_name*.STI. If it does not exist, ICED™ will report an error. (This pattern file can be the same one used to define fill patterns for screen display. The PATTERN command is used to define the pattern file used for screen display.)

The **TEMPLATE** command can be used to display the current LAYER parameters, including pen numbers.

A pattern file may contain definitions of up to 99 patterns in the range 2:100. Pattern 0 is reserved for the no fill pattern. When pattern 0 is assigned to a layer, only outlines are drawn on the plot. Pattern 1 is reserved for a solid fill.

²⁵ Remember that Q:\ICWIN represents the drive and path where you have installed the ICED™. See page 44 for details on the auxiliary file search path.

Assignments of individual plotting patterns to layers are established using the PEN parameter in the LAYER command.

Only layers with non-negative pen numbers are plotted.

For a quick report on a layer definition, type a **LAYER** *layer_id* command with no other parameters.

When the PEN=* option is used in a layer definition, the layer will be plotted using the same pattern number used for screen display as defined by the PATTERN keyword (usually abbreviated to PAT) in the layer definition.

If a layer's pen number is greater than 1 and there is a pattern with the same number defined in the pattern file, components on that layer are filled using the pattern when the plot is printed. If there is no pattern defined for the pen number, the component is drawn only as an outline.

Example:

PLOT PLOTTER=LJET300 PATTERN=SAMPLE ...

See the **PATTERN** command description for samples of some of the fill patterns in the SAMPLE.STI pattern file provided with ICED™.

This example is only a fragment of a complete PLOT command. The plot file created can be printed by the MkPlot utility on a LaserJet printer at 300dpi. The patterns in the file SAMPLE.STI will be used to fill components on layers that have a pen number greater than 1 associated with the layer. Notice that the path name and .STI extension are NOT included in *pattern_name*.

Cell and array outlines are the dotted lines drawn on the boundaries of cells and arrays. The outlines are drawn on the fictitious layer 0. If you want cell and array outlines to be drawn on a plot, you must assign a non-negative pen number to layer 0 with a LAYER command .

SIZE=*size*

You can use the **RdPDF** utility to determine valid paper sizes for your printing device.

The SIZE keyword is not optional. It indicates the paper size on which the plot will be printed. The choices available vary with the plotting device. If you are not sure which sizes are supported by your plotting device, use the FILE → PLOT menu option to generate your plot. ICED™ will list the supported paper sizes automatically when you use the menus.

[DOTS=(1,4) | DITHER]

If your plot command fails with the message "error: *n* layers assigned colors for which for which xxx_DOTS is undefined" you can execute the COLORS.CMD file to assign default definitions. See the **COLOR** command.

This option determines how colors are plotted.

The DOTS=1 mode (the default) prints one dot of color on the page for each pixel of the plot. Only 8 distinct colors can be used in this mode, but the same color can be associated with two different color numbers each with a different level (i.e. priority on a plot). To insure that layers with small but important shapes (e.g. vias) are not obscured, use a high level color and a solid fill pattern. Layers with large shapes that should be in the background (e.g. wells) should use a low priority color and a sparse pattern.

The color used for each layer on the plot depends on LAYER and COLOR commands. The LAYER command assigns a color number to a layer. The COLOR command associates a plot color, screen palette, and level to a color number.

The ONE_DOT parameter of a COLOR command assigns the plot color used when the PLOT command uses DOTS=1. ONE_DOT parameters are initialized by default only for colors 0:7. You must execute COLOR commands or a command file similar to COLORS.CMD to define plot colors for colors 8:15 if you want these colors to be plotted.

Use the **TEMPLATE** command to see current settings of the COLOR and LAYER definitions.

The DOTS=4 mode allows you to use 16 distinct colors on a plot, and each color can be customized. Each pixel in the plot results in a 4-dot "superpixel" on the paper. While each of these dots is one of the predefined 8 colors, the combination of different colors in the superpixel appears to be a new color.

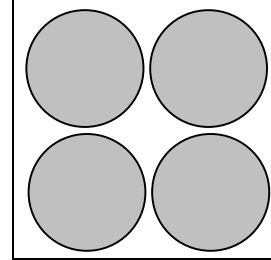


Figure 92:
Superpixel on
DOTS=4 plot.

The colors of the 4 dots in the superpixel for each layer color are defined with the FOUR_DOTS parameter of the COLOR command. These parameters are undefined for all colors by default. Unless you have used a startup command file that contains COLOR commands to define these parameters, you must define them yourself, or use the COLORS.CMD command file to define 16 standard colors. When the PLOT command tries to generate plot data for a color that is undefined, the PLOT command will fail.

When a plot is generated with the DOTS=4 option, each superpixel of the plot takes up 4 pixels of the page; therefore the apparent resolution of the plot is reduced by half. For example, if you use the PLOTTER=LJET300 option, you would see a 300dpi (dots per inch) resolution if you used the DOTS=1 parameter. However, in the DOTS=4 mode, the plot would print at only 150 superpixels per inch. You may need to plot smaller portions of your design to see details, or use the NX and NY parameters to spread the plot over more than one page.

If your design has relatively few layers (8-12), we recommend the DOTS=1 mode. This will give you a higher resolution, allowing you to plot more data on a page. If your design has many layers, using the DOTS=4 mode will provide you with more unique colors to easily differentiate layers.

The DITHER option may be used instead of either DOTS mode when you are plotting on a large format Hewlett Packard color printer/plotter. These devices (especially the HP500/800/5000 series plotters) support a dithering option that allows for more than 8 pure colors. Dithering may change the color of adjacent pixels in the plot. You will need to test your device to see if this option provides more readable plots than the DOTS=1 or DOTS=4 mode.

When DITHER is used, a single pixel will be printed for each pixel in the plot. So the resolution is the same as if DOTS=1 was used.

The 4 colors specified in the FOUR_DOTS parameter of the appropriate COLOR definition are averaged in DITHER mode to define the plot color. The actual color used on a plot also depends on the plotter's dithering algorithms. You may have to experiment to get readable colors with this option.

[LAYERS=*layer_list*]

See **Layer Lists** in the **Syntax Conventions** chapter.

The LAYERS parameter can be used to restrict the layers plotted to those in the *layer_list*. Any components on a layer not included in the list will not be plotted.

[NX=*nx*] [NY=*ny*]

The NX and NY parameters are used to create multipage plots. A larger area can be plotted than will fit on a single page. The pages plotted can then be pasted together to create a large plot of your design. ICED™ will automatically create a .25 inch overlap on each page to make pasting the pages together easier.

NX is used to define the number of pages in the horizontal direction. NY defines the number of pages in the vertical direction. See Figure 93. When the PLOT command is typed in, either NX or NY can be used alone, or they can be used together. The *nx* and *ny* values must be in the range 1:20.

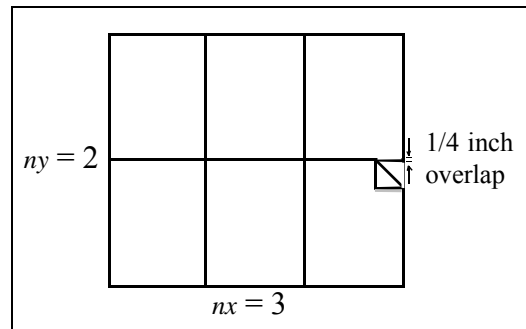


Figure 93: Superpage created by pasting together pages plotted with NX=3 NY=2.

[STRIP= (YES | NO)]

The STRIP keyword is used to limit the length of a plot in the y-direction. The STRIP=YES mode is intended only for roll plotters that support variable length plots (e.g. the HP650C). The PDF file determines the default value for the STRIP keyword for a specific plotter and page size. We suggest that you **always use the default by not including the STRIP keyword** in a PLOT command. The discussion below is for educational purposes only.

This option is not provided on the PLOT menu. It can only be added to typed PLOT commands.

Roll plotters have a fixed width, but a variable length. However, PDF files support only fixed page sizes. Therefore, PDF files for roll plotters define a page size (usually called STRIP) that specifies an excessive length (e.g. 10 feet). This allows you to generate as long a plot as needed. To prevent feet of wasted paper, the STRIP=YES parameter forces the plot software to recalculate the paper length to be exactly long enough to fit the plot data.

The STRIP=NO mode indicates that the fixed paper size (selected by the SIZE keyword in the PLOT command) should be used. This is the default for most plotters and paper sizes.

The paper width is always fixed by the paper size selection. It is not affected by the STRIP keyword.

The use of STRIP=YES restricts the use of the NY keyword. NY=1 (the default) is the only valid use of NY when the strip mode is selected. There is no restriction on the NX keyword that allows a plot to span several sheets of paper in the x-direction.

[R=(0 | 1)]

The plot can be printed in either the portrait or landscape orientations. A portrait orientation means that the page is taller than it is wide. A landscape orientation means that the page is wider than it is tall. The pages in Figure 93 are shown in the portrait orientation.

Using the R=0 parameter in a PLOT command will plot pages in the portrait orientation. No rotation of the design will be performed. Using R=1 will cause ICED™ to rotate the image so it is plotted in the landscape orientation.

It is easier to omit the R keyword entirely. This allows ICED™ to automatically determine which orientation results in the largest scale plot that will fit on the paper.

[FILTER=*filter_size*]

The VIEW
LIMIT
command
controls a
minimum size
filter for the
screen display.

The optional FILTER parameter sets a minimum size in user units (usually microns) for plotable components. *filter_size* must be a real number in the range 0:100.

A bounding box is a rectangle that just contains an entire component. If one side of a component's bounding box is longer than or equal to *filter_size*, then the component will be plotted. If both sides of the bounding box are smaller than *filter_size*, then the component will not appear in the plot.

If the FILTER keyword is not used in the PLOT command, *filter_size* will default to 0 and no components will be filtered.

[DEPTH=*depth*] and [OUTLINE_DEPTH=*out_depth*]

See the
DISPLAY
command sets
the depths for
which cells and
outlines are
drawn on the
screen display.

The optional DEPTH parameter (indicated by CELL DEPTH in the menu) limits the depth in the cell hierarchy for which cell components will be plotted.

OUTLINE_DEPTH (also optional) limits the depth in the cell hierarchy for which the outline of a cell or array will be drawn around the cell contents. If the contents of a certain cell (or array) are not plotted, due to the DEPTH parameter, the value of OUTLINE_DEPTH has no effect.

All components in the root cell are said to have depth 0. Components contained in depth 0 cells are said to have depth 1. Components nested in depth 'n' cells have depth 'n+1'. The cells that form an array of depth 'n' have depth 'n+1'.

The DEPTH parameter limits the depth to which ICED™ will plot components nested in cells. ICED™ will draw only components whose depth is less than or equal to *depth*. *depth* must be a positive integer in the range 0:100. When the DEPTH parameter is not used, ICED™ will draw all nested components.

If DEPTH=0 is used in a plot command, only components which are in the current cell will be plotted. Any components nested in cells or arrays will not be plotted.

OUTLINE_DEPTH works in a similar fashion. *out_depth* must be a positive integer in the range 0:100. Only cell or array outlines with a depth less than *out_depth* (and less than or equal to *depth*) will be added to the plot when the contents of the cell or array are also drawn.

Example:

LAYER 0 PEN 0
PLOT ... DEPTH=4 OUTLINE_DEPTH=1 CELL_LABELS=ON ...

Layer 0 is the layer used to draw cell outlines. If this layer number is defined in a **LAYER** command with the NOPEN keyword or PEN=-1, no cell outlines or labels will be plotted.

The fragment of the PLOT command shown above demonstrates the use of DEPTH, OUTLINE_DEPTH, and CELL_LABELS (covered next). The contents of cells will be drawn if only if they are not nested more than 4 levels deep. The cells added directly to the root cell (these cells have a depth of 1) will be surrounded by a cell outline. A cell outline and a cell label will be drawn instead of the contents for cells or arrays nested 5 levels deep. Cells or arrays nested more deeply than 5 levels will not be indicated on the plot at all.

Depth of cell	Contents drawn	Outline drawn
0	Yes	(no outline to draw)
1	Yes	Yes
2	Yes	No
3	Yes	No
4	Yes	No
5	No	Yes, and a cell label will be added
6 and up	No	No

Figure 94: Results of PLOT...DEPTH=4, OUTLINE_DEPTH=1

The LAYER 0 definition that precedes the example above is added to remind you that this layer must not be set to NO_PEN if you want the cell outlines and labels to be plotted.

[CELL_LABELS=(ON | OFF)]

This option determines whether or not cell labels will be added to a plot in addition to the cell outline when the contents of a cell will not be drawn due to the restrictions of the DEPTH=*depth* parameter. Cells of *depth* +1 will have a cell label drawn instead of the cell contents if CELL_LABELS=ON is used.

[ARRAY_MODE=(BLANK | FULL | SIDES)]

The **ARRAY** command controls how arrays are displayed on the screen.

The optional ARRAY_MODE parameter determines which of the cells that make up an array are plotted. The valid modes are:

FULL	All the cells that make up an array may be plotted.
SIDES	Only the cells on the outside edges of the arrays may be plotted.
BLANK	The contents of arrays will not be plotted.

If the ARRAY_MODE parameter is not provided in the PLOT command, the default mode of FULL will be used.

[FILE=*file_name*]

The PLOTNOW-.CMD file creates the .VEC file in the TMP subdirectory.

The FILE parameter is optional. By default, the plot file name will be *cell_name*.VEC, where *cell_name* is the name of the cell you are currently editing. You can use the FILE keyword to override this default.

You can include a directory path as part of *file_name*. When you do not provide a path, the file is stored in the same directory as the root cell. The file extension is always .VEC. If you include another extension it will be ignored.

[TEXT_ORIENTATIONS=(2 | 4 | 8)]

TEXT_ORIENTATIONS controls how text which has been rotated or mirrored is plotted.

TEXT_ORIENTATIONS=8 means that text has 8 possible plot orientations. All text will be plotted as it is oriented in the cell. Mirrored text will plot as mirrored. Text rotated 180° will plot upside-down and backwards.

TEXT_ORIENTATIONS=4 is rarely used. This option will plot mirrored text in an unmirrored state. Therefore, text will plot in only 4 different orientations: horizontal right-side-up, horizontal upside-down and backwards, vertical forwards (i.e. the letters are read from down to up) or vertical backwards (read up to down).

TEXT_ORIENTATIONS=2 will always plot text as horizontal right-side-up or vertical forwards. This is the default if the TEXT_ORIENTATIONS parameter is not provided in the PLOT command. This provides the most readable plots.

[TEXT_HIGHLIGHTS=(ON | OFF)]

TEXT_HIGHLIGHTS (indicated by TEXT HALOS on some menus) controls how text is plotted. When TEXT_HIGHLIGHTS=ON is used, a small halo of white will be plotted around all text. This allows the text to be visible even in dark areas of the plot.

If the TEXT_HIGHLIGHTS parameter is not provided in the PLOT command, text highlights will not be generated.

[METRIC | ENGLISH]

The METRIC or ENGLISH keywords determine the units of the scale factor for the plot. If METRIC is used, the scale factor will be in microns/user unit. ENGLISH indicates that the scale factor will be in mils/user unit. If you do not

provide either of the ENGLISH or METRIC keywords in the PLOT command, ICED™ will use english units for the plot scale.

(ALL [=scale] | SCALE=scale [[AT] pos] | PRESCALE)

The choice between ALL, SCALE or PRESCALE defines what area of the design is plotted as well as the scale at which it will be plotted. ALL is used when you need to plot all of your data. SCALE is used when you require that only an area of the data be plotted at an exact scale for easy measurements or to insure that several different plots all use the same scale. PRESCALE is the easiest way to plot an area of your design. The scale will be calculated to allow the selected data to fit on your page.

ALL will plot the entire design. If present, *scale* defines the scale of the plot. If *scale* is not used, ICED™ will calculate the scale required to fit the data on one page (or superpage if the NX and NY keywords are used). If *scale* is provided, the center of the data will be located at the center of the page and the drawing will be cropped at the edge of the page if the data will not fit. The units of the scale factor are determined by the ENGLISH or METRIC keywords.

Example:

PLOT PLOTTER=HP650C PATTERN=NONE SIZE=D ENGLISH ALL=125

This plot command will plot the entire design on a single page on a Hewlett Packard 650C printer. The scale used will be 125 mils/user unit (1/8"/user unit). If the design data will not fit on the page, it will be cropped at the edges of the page. The center of the design will be at the center of the page. Since the pattern file NONE.STI was specified, no fill patterns will be used.

Using the SCALE=*scale* parameter will plot as much of the data as will fit on your page at the scale specified. If you provide the *pos* parameter directly after the SCALE=*scale* parameter, that position will be used as the center of the plot. (The AT keyword makes the command more readable, but it is not required.) If you do not provide a center position, or if you use the menu to generate the PLOT command, you define the boundaries of the plot with the mouse. A plot window will appear on the screen to allow you to select the area to be plotted. The rectangle represents the size of your paper at the chosen scale of the plot.

The PRESCALE keyword is the most useful way to plot a portion of your design. When you use PRESCALE, ICED™ will allow you to indicate with the mouse the area your design you wish to plot. Then ICED™ calculates the scale required to allow the data to fit on the page size you have selected. You will be prompted with the calculated scale, allowing you to override it if you desire. To use the calculated scale, simply hit <Enter> at the prompt. You then select the exact area to be plotted using the mouse to position the plot window rectangle.

Keyword Order

If you type a PLOT command, it is best to use the PLOT keywords in the order indicated in this chapter. The PLOTTER keyword must be first. The PATTERN keyword comes next, followed by the SIZE keyword. The exact order of the other keywords is not critical, except for the ALL, SCALE, or PRESCALE choice of keywords, which must be the last parameter in the command.

Creating Bitmaps

This version of ICED™ supports the creation of bitmap graphics files. These bitmap files can be added as graphics into documents as needed.

To create a bitmap file you follow the same steps that you would to plot your layout on a plotter. If you have not already read the preceding information on the PLOT command you should do so before attempting to create a useful bitmap graphic.

When you execute the FILE->PLOT menu option to create your bitmap, there are at least two choices for bitmap export. At the first prompt for the PLOTTER value, select either of the following options:

BMPCOLOR for color bitmaps

BMPMONO for black and white bitmaps

Use the **PATTERN** command to see the name of the current screen display pattern set.

At the next prompt for PATTERN, select the pattern file you wish to use for fill patterns. Use the pattern file currently selected for the screen display if you want to see the same patterns in your bitmap. The actual pattern used for displaying each layer in the bitmap is determined by the PEN parameters of the LAYER definitions. **Unless PEN=* is used for each layer, the patterns you see in the bitmap may be different than the patterns on the screen.**

When prompted for SUPER-PIXEL MODE, we recommend that you always use the 1 DOT mode for a single dot of color for each pixel of the plot. If you must use the 4 DOT option to achieve more than 8 colors, read the description of the DOTS keyword on page 272.

To add custom sizes of bitmaps to a plotter definition file, see the **MkPDF** utility description.

The sizes supported in your selected bitmap plotter definition file are listed next. The distribution copies of the BMPCOLOR.PDF and BPPMONO.PDF plotter definition files allow you to choose from the following sizes:

AA	3.75 inches x 1.691 inches	147.6 mm x 66.6 mm
A	7.5 inches x 3.83 inches	295.3 mm x 133.2 mm

Selecting NONE at the prompt for more options will plot your entire layout in the created bitmap.

To create a PLOT command to create your bitmap without using the menu, you should use a command similar to:

Example:

**PLOT PLOTTER="BPPMONO" PATTERN="SAMPLE" SIZE=A ...
.. ARRAY_MODE=FULL DOTS=1 ALL**

This command will create a file *cellname.VEC* which can be used to create a black and white bitmap of your layout. (We'll show you how to create the *cellname.BMP* file in a moment.)

All colors will plot as black, and the bitmap will have a white background. You will need to respond with <Enter> at the plot scale prompt since the scale is not provided in the command. Responding by typing <Enter> will use a default that will fit your entire cell in the bitmap.

After you have executed the PLOT command above and it reports that the *cellname.VEC* file has been created, you must run the MkPlot utility to create the bitmap file. This is the step where you determine the resolution of the bitmap.

To execute the MkPlot utility, you must open an MS-DOS console window. The easiest way is to type the SPAWN command (or use the 3:SPAWN menu option), then type the following command in the **new console window (not the layout editor prompt)**:

Example: **MKPLOT *cell_name***

where *cell_name* is the name of your cell.

The only prompt is for the resolution of the bitmap file. Word processors face serious problems dealing with high-resolution bitmaps. They must display the bitmap on a monitor with 50 to 150 dpi resolution. Then they must print the bitmap on a printer with anywhere from 75 to 720 dpi resolution. In general neither the screen nor printer resolution will match the nominal resolution of the bitmap. You can make it easier for your word processor by using a low bitmap resolution that is also a sub-multiple of your printer's resolution.

Examples of good choices for bitmap resolution for the following printers are listed at the right.

Printer	Resolution
HP	100
Epson	120

When the MkPlot command is complete, the bitmap file is created.

You can leave the console widow open if you want to adjust aspects of the bitmap by executing LAYER, COLOR, and PLOT commands in the layout editor window. Simply keep repeating the MkPlot command in the console window after each PLOT command is complete in the layout editor window. When you are happy with the final bitmap, close the console window by typing the EXIT command in the console window

PROTECT and UNPROTECT *Prevent changes to components or layers.*

[UN]PROTECT ALL

[UN]PROTECT LAYERS=*layer-list*

PROTECT

PROTECT SELECT

The **BLANK** command also prevents components from being selected and makes them invisible.

The **PROTECT** command is used to prevent modification of components. Any component that is protected cannot be selected. This helps prevent accidental changes to specific components. For example, the **PROTECT** command can be used when you want to avoid making any changes to a semi-custom base layout.

You will not be able to select a protected component. The **SELECT** command, or any command which has an embedded **SELECT** option, will not be able to work on a protected component. This applies even to commands which do not modify geometry (e.g. the **SHOW** command). Protected cells and arrays cannot be selected by commands, but protected cells can still be edited by name (if they are not stored in a view-only cell library).

[UN]PROTECT ALL

The **[UN]PROTECT ALL** command, unprotects or protects all unblanked components in the drawing.

[UN]PROTECT LAYERS=*layer_list*

See **Layer Lists**
in **Syntax**
Conventions.

The [UN]PROTECT LAYERS command, unprotects or protects all unblanked components that are on the layers in *layer_list* at the time the command is executed. The list of all protected layers is retained by ICED™. Components added to protected layers after executing the PROTECT command are **not** automatically protected, but the form of the PROTECT command without any parameters can be used to protect all new components on layers which have been protected.

Example:

PROTECT LAYER 0

This use of the PROTECT command protects all cells in the drawing. You will not be able to select any cells added to the current cell prior to the PROTECT command. However, cells added to the current cell **after** this command is executed will not be protected.

PROTECT

When the PROTECT command is used without any other parameters, it protects all **new** components on protected layers added since any PROTECT LAYERS commands were executed. ICED™ keeps track of which layers in the current cell are protected.

Example:

PROTECT LAYER M1
ADD WIRE LAYER=M1
PROTECT

The second PROTECT command will protect the new wire component on layer M1.

PROTECT SELECT

The PROTECT SELECT command, protects all fully selected components. Partially selected components are unaffected. The components must be selected **before** executing the PROTECT SELECT command.

Interactions Between the UNPROTECT and SELECT Commands

If you issue a SELECT NEW command just after an UNPROTECT command, any newly unprotected components will be selected.

QUIT

Terminate the edit session without saving the current cell.

QUIT [YES | NO]

Refer to the **EXIT** command for a chart comparing the four termination commands.

The QUIT command can be used to return from editing the current cell without saving changes made to it. If changes were made to the geometry in the cell, you will be warned that work will be lost, then given the option to cancel the command. When you are only browsing a cell, using QUIT instead of EXIT will prevent ICED™ from overwriting the cell file and the cell backup file.

When QUIT is used to return from editing a subcell you entered using the EDIT, P_EDIT, or T_EDIT commands, the changes made to the subcell will not be flagged for saving, and the cell file will not be saved when the ICED™ session is terminated.

See **Journaling and Data Recovery** in **Appendix B** if you need to recover unsaved work.

When QUIT is used while editing the root cell (the cell edited when you first launched ICED™), then the editor terminates. The cell file for the root cell is not saved. If you edited other cells during the current session, their cell files **are** saved before the editor terminates if they were flagged for saving by the EXIT or LEAVE commands.

The optional YES or NO keywords can be used to avoid ICED™ presenting you with the following prompt when changes have been made to the geometry in the cell.

All work on cell CELLNAME will be lost --
Do you really want to QUIT (Y or N)?

Using the **JOURNAL** command to terminate the layout editor will prevent any cell files from being saved.

The YES or NO keyword will be used to answer the prompt before it is asked. This is useful when the QUIT command is used in a command file. If you use QUIT NO, ICED™ will not perform the QUIT command if geometry has changed. (Changes to the view window or to the selection status of components are not considered changes to the geometry.). An error is reported and the remainder of the command file is not processed.

REDRAW

Refresh the screen.

REDRAW

The REDRAW command is used to refresh the screen.

Example: **RED**

Occasionally the layout window may be left partially blank when switching between windows. The abbreviated command above will force the layout to be redrawn.

This command is also useful in command files that call DOS programs that leave data on the screen.

Any time the redraw of a dense layout is taking too long, you can press <Esc> to stop the redraw. If you want to continue drawing the display where it was interrupted, simply issue another REDRAW command as the next command. Otherwise, you can simply enter any other command to continue without the view window being completely redrawn.

RESOLUTION

Define the grid for digitizing points.

RESOLUTION [STEP=*step_size*] [[MODE=] (SOFT | HARD)]

The RESOLUTION command is used to define the grid used when digitizing coordinate data.

When the mouse is used to enter coordinates, they will be rounded to the nearest resolution grid point. The mouse cannot be used to digitize a point which is not a point on the resolution grid.

The resolution grid affects only points digitized with the mouse or (when the resolution mode is set to HARD) vertex points created by commands which calculate points, like ADD CIRCLE. If you type in coordinate data yourself using a command like **ADD BOX AT (0.05, 0.05) (10.05, 12.05)**, the points will **not** be rounded to lie exactly on the resolution grid.

See the **SNAP** command for defining a temporary grid.

The resolution grid is technology dependent and it rarely needs to be altered. The RESOLUTION command is most commonly used in the startup command file that ICED™ executes when you create a new cell. If you prefer to have points temporarily snap to a grid of your own choosing, use the SNAP command.

See the **GRID** command to display grids on the screen.

Changing the resolution grid with the RESOLUTION command will also reset the STEP and OFFSET parameters set with the SNAP command. Changing the resolution grid of a design will **not** change any existing geometry.

The resolution grid is not shown on the screen. However, the GRID command, which defines the display grids shown on the screen, can be used to display the same grid as the resolution grid.

[STEP=*step_size*]

See the **NDIV** command line parameter to learn more about database units.

The *step_size* parameter determines the space in user units between points on the resolution grid. The step is the same in the X and Y directions. It must be a real number which will be rounded to an integral number of database units.

Remember that:

1 database unit = 1 user unit / NDIV (where commonly NDIV=1000).

Example:

RES STEP=0

Since ICED™ rounds the step size to the nearest legal value in database units, this command will set the resolution grid to one database unit. This the finest grid possible since coordinates are stored as multiples of database units.

[[MODE=] (SOFT | HARD)]

The MODE keyword controls rounding of vertices calculated by commands like ADD or CUT. If MODE=HARD, vertex points calculated by commands like ADD CIRCLE will be forced to be exactly on the resolution grid. If MODE=SOFT, calculated points can be off-grid. This can be desirable if post-processing software allows points to be off-grid and will perform rounding of point coordinate data to conform with the technology requirements.

Keyword Order

The order of the STEP and MODE keywords is unimportant. If one of the keywords is absent, its value will not change.

Example:

RES

This shorthand for the RESOLUTION command, used without any parameters, reports the current settings of the resolution and snap grids.

ROTATE

Move components by rotating about a point.

ROTATE *rot_code* [[AT] *axis_pos*]

See **COPY**
ROTATE to
rotate
components and
retain the
originals.

The ROTATE command relocates all fully selected components by rotating them around a point axis. The rotation is restricted to multiples of 90 degrees.

If no components are selected prior to executing the ROTATE command, an embedded SELECT NEAR command is generated to allow you to fully select a component. If you wish to rotate several components, select them with the SELECT command before executing ROTATE. Partially selected components are unaffected by the ROTATE command.

rot_code

The required *rot_code* parameter indicates the number of counter-clockwise 90° rotations to perform. Legal values for the *rot_code* parameter are listed in Figure 95. This is the same table as that provided in the COPY ROTATE command description.

<i>rot_code</i>	counter-clockwise rotation	result
0	0°	F
1	+ 90°	⌞
-1	- 90°	⌞
2	+180°	⌞
-2	-180°	⌞
3	+270°	⌞
-3	-270°	⌞

Figure 95: ICED™ Rotation codes.

[[AT] *axis_pos*]

The *axis_pos* parameter defines the point about which the components will be rotated. It should be a coordinate pair of real numbers. The AT keyword is not required, it is provided to make the command more readable.

Example: **ROTATE 1 AT (10.5, 20)**
 ROT-3 10.5 20

Both commands rotate all fully selected components by 90 degrees counter-clockwise about the point (10.5, 20). Note that the parentheses, comma, and the AT keyword are all optional.

If you do not specify the axis position as part of the **ROTATE** command, or if you enter the command from the menus, you will have to enter the axis position with the mouse. The following command expects mouse input:

Example: **ROTATE 2**

This command will allow you to select the axis position with the mouse. All fully selected components will be rotated 180° about the point chosen with the mouse.

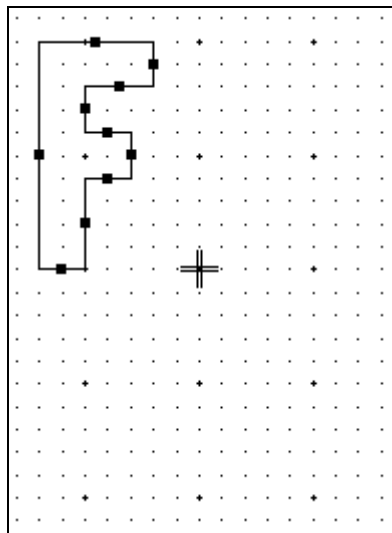


Figure 96: Using the cursor to define *axis_pos* in the ROTATE command.

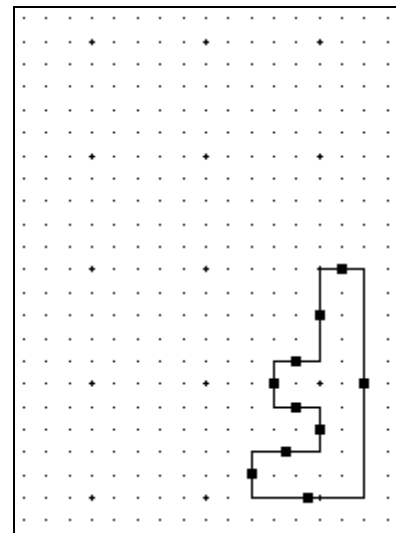


Figure 97: Result of ROTATE in Figure 96.

RULER

Measure the distance between two points.

RULER

The RULER command is used to measure the distance between two points.

You must use the mouse to define the coordinates of the two points. After you have entered the RULER command, a full screen crosshair will appear. Place it over the first point and press the left button. As you move the crosshair, the position of the starting point, the current crosshair location, the diagonal distance between the two points, and the x and y displacements are displayed on the screen. Pressing the left mouse button digitizes the second point and ends the command. The coordinate and distance information will remain on the screen until you enter another command.

See the **RESOLUTION** command to learn about the resolution grid.

The snap grid and the value of the cursor snap mode are ignored during the ruler command. The cursor will snap to points on the resolution grid. Since only points on this grid can be digitized with the mouse during the RULER command, it can be used only to measure distances between points on the resolution grid.

The **CURSOR** command sets the snap mode that controls how the cursor snaps to the snap grid during other commands. The **SNAP** command sets the snap grid.

SELECT and UNSELECT

Select or unselect components.

```
[UN]SELECT [ IDS (=id_list | BEFORE id_int | UPTO id_int | AFTER id_int ) | ...
    [ TAGS (=0 | =* | =id_list | BEFORE tag_int | UPTO tag_int | AFTER tag_int ) ]...
    [ LAYERS=layer_list ] ...
... [ COMPONENT | BOX | POLYGON | WIRE | LINE | TEXT ...
    ... | CELL=cell_name | CELL=* ...
    ... | ARRAY=cell_name | ARRAY=* ...
    ... | SIDE | END ] ...
... [ NEW | PARTS | FULL ] ...
... [ ALL | IN pos1 pos2 | AT pos | NEAR pos ]
```

```
[UN]SELECT [PUSH | POP | EXCHANGE | FAIL]
```

```
SELECT LIST [ # ]list_name [ NEXT | PREVIOUS | FIRST | LAST ]
```

Manipulations in ICED™ are generally performed on selected components. The SELECT and UNSELECT commands are used to select the components you wish to manipulate. You do not have to select all the components you wish to manipulate with a single command. You can use a combination of SELECT and UNSELECT commands to build up an arbitrary set of selected components.

The <Shift> select feature of embedded SELECT commands is not enabled when SELECT commands are entered directly.

Some ICED™ commands distinguish between fully and partially selected components. A box, polygon, wire, or line is partially selected if only some of its parts are selected. The same component is fully selected if all of its parts are selected. Cells, arrays, and text components are always either fully selected or unselected.

You can select any or all of the sides of a box or polygon. Similarly, you can select any or all of the segments of a wire or line and either or both ends.

If you use SELECT commands in a command file, use the **XSELECT** command to avoid problems if nothing is selected.

ICED™ marks the selected boxes, polygons, wires, and lines by drawing select marks (small white boxes) on their selected parts. ICED™ marks selected cells, arrays, and text by drawing select marks on the sides of their bounding boxes. (A bounding box is a box that just surrounds a component.)

There are three basic classes of [UN]SELECT commands:

- the normal [UN]SELECT commands that use a variety of criteria to identify the correct components (see below),
- the [UN]SELECT stack commands that use the PUSH, POP, EXCHANGE or FAIL options to save and restore a set of selected components, or to select components that caused the previous command to fail (see page 304), and
- the SELECT LIST commands that select components one by one from a named list (see page 306.)

Normal SELECT Commands

In general, the options for a normal [UN]SELECT command provide for up to five tests that restrict the components it acts on. They are called the id/tag, layer, category, status, and method tests.

[IDS (=id_list | BEFORE id_int | UPTO id_int | AFTER id_int)

or

TAGS (=0 | =* | =id_list | BEFORE tag_int | UPTO tag_int | AFTER tag_int)].

The DRC program adds a tag number to each shape it creates to mark layout rule violations.

You can add one ID test or TAG test to a SELECT or UNSELECT command. The IDS keyword restricts components selected by their component ids, unique positive integers assigned consecutively as the components are added to the design. The TAGS keyword is used to (un)select components by the tag number added by the DRC to shapes on error layers.

When used, the id test or tag test should be the first parameter in the SELECT command.

The **SHOW** command will report the unique ids of selected components.

Using **IDS=id_list** allows you to enter a list of component id integers. To select a single component by its id, use "SELECT ID=*id_int*", where *id_int* is an id integer. This form of the SELECT command can be useful in selecting one of two copies of a component that are on top of each other.

To restrict the SELECT command to a range of component ids, use one of the id keywords. **BEFORE id_int** will restrict selection to all components with an id less than *id_int*. **UPTO id_int** will allow only components with an id less than or equal to *id_int* to be selected. To restrict selection to those components with an id greater than *id_int*, use **AFTER id_int**.

Example:

SELECT IDS AFTER 9
UNSELECT IDS AFTER 20

The combination of commands above will select components with ids in the range 10:20.

A tag number is the rule number that generated an error shape. Look at the DRC rules compiler log file to correspond a tag number to a specific rule.

Use a tag test to select shapes by the tag numbers used by the DRC (Design Rules Checker, a separate product available from IC Editors.) The syntax for a tag test is very similar to that of the id test. You can specify a single tag number in *tag_list*, or all tags before, after, or up to a certain tag number. In addition you can use **TAGS=0** to (un)select components that have no tag number (i.e. components not created on error layers by the DRC), or **TAGS=*** to (un)select all components with non-zero tag numbers.

[**LAYER=layer_list**]

See **Layer Lists** in **Syntax Conventions**.

The optional layer test starts with the **LAYER** keyword. It confines the effects of the command to certain layers. For the purposes of [UN]SELECT commands, cells and arrays are on layer 0. If the **LAYER** keyword is not used, components on any layer may be (un)selected.

[**COMPONENT** | **BOX** | **POLYGON** | **WIRE** | **LINE** | **TEXT** | **CELL=cell_name** ...
 ... | **CELL=*** | **ARRAY=cell_name** | **ARRAY=*** | **SIDE** | **END**]

The category keyword restricts the type of components that can be (un)selected and determines whether the command acts on entire components or only on their parts. There can be at most one category keyword in the command. If no category keyword is used, the COMPONENT keyword is used as the default. In this case, the command can fully (un)select entire components of any type.

Category Keyword	Components Affected
COMPONENT	Entire components of any type (the default if no category keyword is supplied)
BOX	Entire boxes
POLYGON	Entire polygons and boxes
WIRE	Entire wires
LINE	Entire lines
TEXT	Text
CELL= <i>cell_name</i>	Cells, or arrays of cells, named <i>cell_name</i>
CELL=*	Any cell or array
ARRAY= <i>cell_name</i>	Arrays of cells named <i>cell_name</i>
ARRAY=*	Any array
SIDE	Individual sides of polygons or boxes; individual segments and the ends of wires and lines; and whole cells, arrays, and text components
END	The ends of wires and lines

Figure 98: The SELECT command category keywords

The SIDE and END keywords are the only keywords that allow you to partially (un)select a component. However, they behave very differently. The END keyword will only allow you to (un)select the ends of wires or lines; no other components will be affected. The SIDE keyword allows you to select any component or any part of a box, polygon, wire, or line. It differs from the COMPONENT keyword in that it allows you to (un)select individual parts of components.

Notice that the CELL keyword allows you to (un)select arrays, but the ARRAY keyword does not allow you to (un)select cells. Similarly, the POLYGON keyword allows you to (un)select boxes, but the BOX keyword does not allow you to (un)select polygons. (Remember that ICED™ will automatically turn rectangular polygons with horizontal and vertical sides into boxes.)

[NEW | FULL | PARTS]

The optional status test can restrict components to (un)select by their current select status. Only one status keyword can be used.

The NEW keyword selects components that have been added or changed by the last command. Consult the command descriptions for information about what components are marked as 'new'. The NEW keyword cannot be used in an UNSELECT command. The [UN]SELECT command marks as 'new' any component whose select status was changed.

The FULL keyword restricts the components considered to those already fully selected. Therefore, SELECT FULL makes no sense, since the components are already selected. However, UNSELECT FULL unselects only fully selected components but leaves all partially selected components unaffected.

The FULL keyword is incompatible with the END and SIDE category keywords.

The PARTS keyword allows you to (un)select partially selected components. A SELECT PARTS command fully selects partially selected components. UNSELECT PARTS unselects partially selected components. In either case, the result is fully selected or unselected components.

The PARTS keyword is incompatible with the following SELECT category keywords: SIDE, END, CELL, ARRAY, and TEXT. The first two are used to partially (un)select components, so since PARTS indicates that components should be fully (un)selected, the keywords are incompatible. The other categories indicate components that can not be partially selected, so a SELECT command with a PARTS keyword would never (un)select them.

[**ALL** | **IN** *pos1 pos2* | **AT** *pos* | **NEAR** *pos*]

The method test provides the final restriction on the components that are to be (un)selected. The selection method is either to restrict components selected by their positions, or to select all eligible components. There can be only one method keyword in each [UN]SELECT command.

The ALL keyword (un)selects all components which pass the other four tests.

The IN, NEAR, and AT keywords (un)select components on the basis of their positions. The IN keyword requires two positions that define the corners of a select box. The NEAR and AT keywords each require one position. If you enter the [UN]SELECT command from the keyboard (or a command file) you can include the numerical values of these coordinate positions as part of the command. However, you usually digitize the necessary positions with the mouse.

The IN keyword (un)selects components (or parts of components) that are entirely inside of or cross the edge of the select box. Use the bounding box of a cell, array, or text component to (un)select it. See Figure 99 for an example of proper use of the select box to select a cell. Figure 100 shows that the cell is successfully selected.

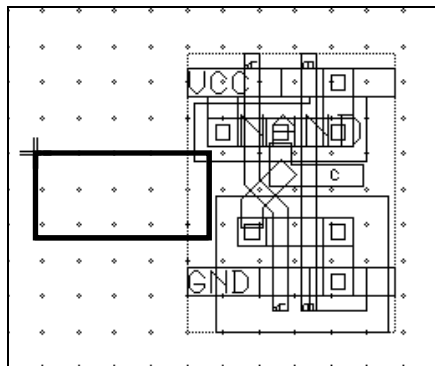


Figure 99: The select box intersects the bounding box of the cell.

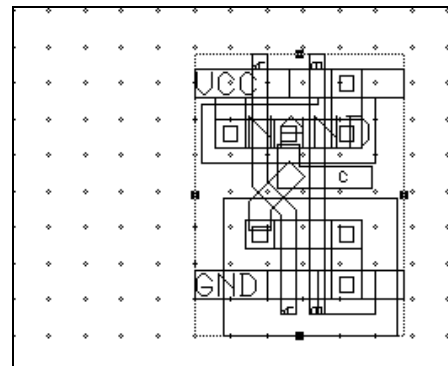


Figure 100: The cell is selected.

The select box does not select components that completely surround it. Figure 101 and Figure 102 demonstrate what happens when the select box is inside a cell's bounding box, but does not intersect it.

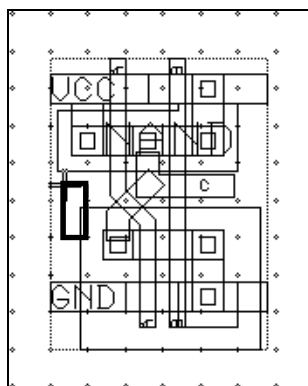


Figure 101: The select box is completely inside the cell.

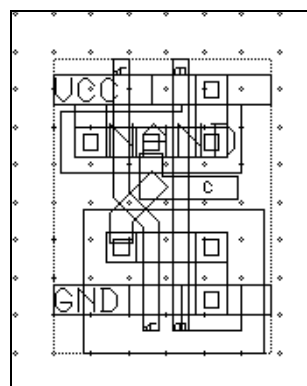


Figure 102: The cell is not selected.

If the component and the select box share a common edge, the component (or component part) is (un)selected if the select box is inside the component's bounding box. This behavior is designed to allow you to select either of two components that share a common side. Figure 103 and Figure 104 demonstrate this use of SELECT IN.

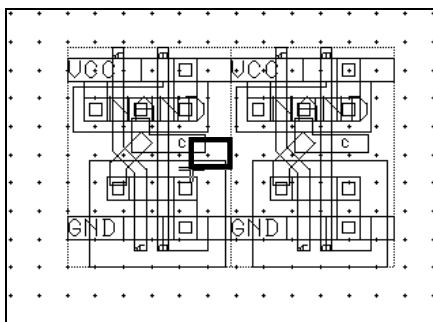


Figure 103: The select box and both cells share edges.

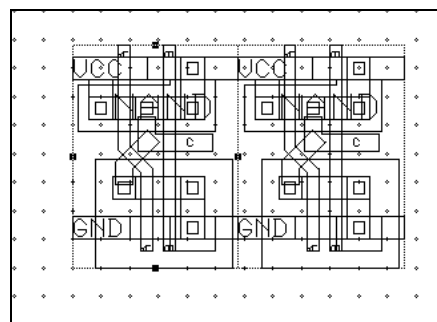


Figure 104: Only the cell on the left is selected since the select box is inside that cell.

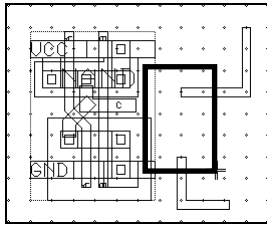


Figure 105: The select box intersects the cell and two wires.

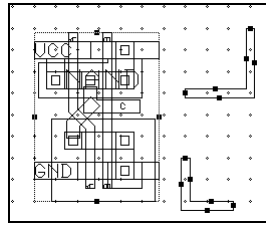


Figure 106: If **SELECT IN** is used, the cell and both wires are fully selected.

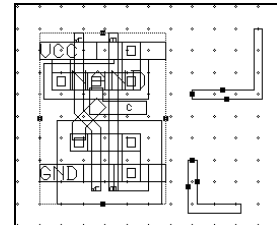


Figure 107: If **SELECT SIDE IN** is used, the cell is fully selected and the wires are partially selected.

These figures demonstrate how the same select box will provide different results depending on the use of other keywords. If **SELECT CELL * IN** had been used, only the cell would have been selected. If **SELECT WIRE IN** had been used, only the wires would have been fully selected.

See the **NEAR** command to set the size of the near box.

The **NEAR** keyword (un)selects components in much the same way as the **IN** keyword. However, the **NEAR** keyword only requires that you specify the center of the box used to select components. This box is called the near box. The size of the near box is computed from the size of the current view window using parameters set by the **NEAR** command.

The **AT** keyword can only be used to (un)select cells, arrays, or text. It (un)selects these components if the **AT** position is inside the component's bounding box. It does not (un)select a component if the **AT** position is on the edge of the bounding box.

Examples of Combinations of Keywords

Example: **SELECT LAYER WELL IN (0, 0) (10, 10)**

This command selects all components on layer WELL that are located inside of, or cross a box with corners at (0, 0) (10, 10). Since layer 0 is not on the layer list, cells and arrays will not be selected.

Example: **UNSELECT LAYER -WELL ALL**

The use of '-' in the layer list means that all layers are chosen except for the indicated layer. This command unselects all components that are **not** on layer WELL.

Example: **SELECT LAYER M1+POLY END IN**

This use of the LAYER keyword selects the ends of wires and lines on layers M1 and POLY that are inside of the select box. Since the select box coordinates were not given as part of the command, they must be digitized with the mouse.

Example: **SELECT LAYER 0+5:50-10 IN**

This command selects cells, arrays, and components on layers 5 through 9 and 11 through 50 that are entirely within or cross the select box digitized with the mouse.

Example: **SELECT POLYGON ALL**

This command selects all polygons and boxes in the drawing.

Example: **SELECT CELL NAND AT (5, 5)**

This use of the CELL keyword will select a copy of cell NAND or an array of NAND cells whose bounding box contains the point (5, 5). It does not select the cell or array if the point (5, 5) is on the edge of the bounding box.

Example: **UNSELECT PARTS ALL**

This command unselects all partially selected components but leaves fully selected components unaffected.

The SELECT Stack Commands: [UN]SELECT [PUSH | POP | EXCHANGE | FAIL]

The SELECT Stack commands remember a set of selected components. This memory is implemented as a stack of depth one. The PUSH, POP, and EXCHANGE operations record and retrieve a set of selections. Only one set can be recorded. Every time ICED™ executes a PUSH operation, any information saved by an earlier PUSH operation is lost.

You can save a named set of components with the LIST command. This is explained on page 306.

The UNSELECT keyword does not have the same result in stack select commands that it does in normal select commands, so read the command descriptions carefully.

The SELECT PUSH command saves the current select information on the stack.

The UNSELECT PUSH command saves the current select information on the stack and then unselects all components in the drawing.

The SELECT POP command selects the components on the select stack. Any components that were already selected remain selected as well.

The UNSELECT POP command starts by unselecting all components. Then it selects the components on the select stack.

The UNSELECT EXCHANGE and SELECT EXCHANGE commands are identical. They both save the current select information on the stack, unselect all components in the drawing, and then select only the components that were on the select stack at the beginning of the command.

The SELECT FAIL and UNSELECT FAIL commands are identical. They are used to select the components that caused the last command (most often a MOVE command) to fail. The SELECT FAIL command starts by performing a SELECT PUSH operation. Then it unselects all components except those that

caused the previous command to fail. This allows you to examine and manipulate the troublesome components. When you are done, you can use UNSELECT POP to restore your original selections.

Using the SELECT Stack Commands on Large Databases

If you are editing a large, flattened layout, you may want to avoid using the SELECT stack commands. When doing any command that involves manipulating the select stack, ICED™ must look at every component to see if its select stack status needs updating.

This is generally not a problem except for large flat designs that require page swapping. If you are editing such a design, or creating command files that are to be used with such a design, using UNSELECT ALL instead of UNSELECT POP for example can save quite a bit of time.

Many command files that operate on selected components will run quite a bit faster on large databases if you add a few lines similar to those below to execute the SELECT STACK commands only if necessary.

Example:

```

IF(%N.SELECT!=0){
  UNSELECT PUSH
  #POP.FLAG=1;
}
.
.      ! rest of command file not shown
.
IF(%POP.FLAG!=0) UNSELECT POP
ELSE UNSELECT ALL

```

SELECT LIST Commands:

SELECT LIST [#]*list_name* [NEXT | PREVIOUS | FIRST | LAST]

See the
Command File
Programmers
Reference
Manual to learn
more about
macros.

The LIST command stores a list of the id numbers (unique numbers added consecutively to all components) in a named list. You can then select components one at a time from this list with SELECT LIST commands.

The list of ids is stored in an ICED™ macro (similar to a variable in other programming languages.) The '#' symbol is used in many commands to indicate the name of a macro. While it is not required in a SELECT LIST command, it is highly recommended to prefix the *list_name* with a # to avoid confusion.

Example:

```
SELECT ALL  
LIST GLOBAL #MYLIST  
UNSELECT ALL  
SELECT LIST #MYLIST NEXT
```

This LIST command in the example above stores the id numbers of all components in the current cell. The SELECT LIST command can be repeated as many times as necessary to select each component one at a time.

The ids numbers are stored in the list in sorted order. The first SELECT LIST NEXT command (or the SELECT LIST FIRST command) will select the component with the lowest id number. The next SELECT LIST NEXT command will then select the next lowest id component, etc. Components deleted, cut, or merged since the LIST command was executed are skipped over.

You can traverse the list in reverse order with the LAST and PREVIOUS keywords.

Lists are destroyed when you exit the cell in which they created.

SHOW

Display component or macro data.

```
SHOW [ PROGRAM[=code]] ...  
    ... [ SELECT | ALL ] ...  
    ... [ LAYERS=layer_list ] ...  
    ... [ NOLIBS_ | ROOT | LIBS=( NONE | ROOT | EDITABLE | ALL ) ]...  
    ... [ WIRE_OUTLINES ] ...  
    ... [ AREA_ONLY | COMMANDS_ONLY ] ...  
    ... [ IDS | NOIDS ] ...  
    ... [ BY_NUMBER | BY_NAME ] ...  
    ... [ MACROS=match_string | USER_MACROS=match_string ...  
        ... | SYSTEM_MACROS=match_string ] ...  
    ... [ LIST=list_match_string ] ...  
    ... [ LINE_LENGTH=length ] ...  
    ... ( FILE=(file_name | *) | APPEND)
```

See the
Command File
Programmers
Reference
Manual to learn
about program
macros often
used to supply
values in
command files.

The SHOW command will report the data for selected components or for program macros. This data can be saved in a file or displayed on the screen. The data is generated in command format (i.e. the data for each component is generated as a command that would create exactly the same component at the same location if executed).

The data generated by the SHOW command for a component includes:

- the type of component,
- the cell name (if the component is a cell),
- the layer the component is on,
- the unique id of the component,
- the location of the component,
- and
- other data which varies with component type.

In any case, the data provided is in the form of ICED™ commands that can create identical components.

When the SELECT keyword is used, the SHOW command can also report component area, perimeter length, and wire length if applicable. This area and length data is provided in comment format (preceded with an exclamation mark "!") so that if the SHOW command creates a file, the commands in the file will still be executable even though the non-executable reports of area and length are in the same file.

Macros store values. When used in a command, a macro string is replaced with a value before the command executes.

Other keywords of the SHOW command report current macro settings (MACROS, USER_MACROS, or SYSTEM_MACROS). The user-defined macros are reported in command format so that identical macros will be created by executing the command file created by the SHOW command in any future ICED™ session. System-defined macros are reported in comment format so that they will not be executed by a command file created by the SHOW command.

[PROGRAM[=*code*]]

When this parameter is used, it must be the first parameter in the command.

The PROGRAM keyword is used to create data in a fixed format suitable for input to another program. The optional *code* parameter is intended to format the data in the syntax of older versions of ICED. This allows programs written to parse SHOW files to be compatible with newer versions of ICED without modification. Using the *code* parameter will result in the command parser of the older version of ICED being used, so use the SHOW command syntax consistent with that version.

<i>code</i>	Version
0	2.xx
1	3.xx

Figure 108

Use of the PROGRAM keyword will also result in program information in comment format being added to the SHOW report. The information added includes:

- ICED version number,
- how many database units are in one user unit (NDIV),
- the name of the root cell,
- the resolution grid step size,
- and
- the list of named layers with the default wire size of each layer.

[SELECT | ALL]

The SELECT keyword will cause the SHOW command to report on only selected components. The components can be selected prior to executing the SHOW command.

See the
SELECT and
NEAR
command
descriptions.

If no components are selected prior to executing the SHOW SELECT command, ICED™ will execute an embedded SELECT NEAR command to allow you to select a component to report on. ICED™ will display a cursor with a small box attached to it. Position this cursor on a boundary of the desired component and click the left mouse button. All components with sides within the boundaries of the near box will be selected.

Using the ALL keyword will result in a report on all components in the design **as well as the components inside nested cells** as controlled by the LIBS keyword. (When no LIBS keyword is used, the use of the ALL keyword causes LIBS=ALL to be added to the command. See the next page.)

[LAYERS=*layer_list*]

See page 106 to
learn details of
the syntax of a
layer list.

Use this parameter to restrict the show report to components on layers in the *layer_list*. Unless the SELECT keyword is also used in the same command, the keyword ALL is added automatically when a LAYERS parameter is defined in the SHOW command. The keyword LIBS=ALL is also added automatically when the LAYERS keyword is used (unless a NOLIBS, ROOT or LIBS keyword is also used in the SHOW command.)

Example: **SHOW LAYERS=PDIF+NDIF**

This example will display information on all components on the PDIF and NDIF layers. Whether or not components are selected when the command executes is irrelevant. Components in nested cells on these layers are included in the report.

[NOLIBS | ROOT | LIBS=(NONE | ROOT | EDITABLE | ALL)]

These keywords are used only with the ALL keyword. (When the SELECT keyword is used instead, then LIBS=NONE is used automatically.) They control how nested cells will be handled. Nested cells from the indicated libraries will generate lines in the SHOW report which can create identical cells (and cell files), including an ADD command for each component in the nested cell. Cells in other libraries will generate only a single ADD command for each reference, no information on the components nested inside of it will be included, and equivalent cells will not be created if the SHOW file is executed.

The NOLIBS keyword is a synonym for LIBS=NONE. The LIBS=NONE parameter will prevent any nested cells from having their components listed. Each cell nested in the current cell will generate a single ADD command.

The
ICED_PATH
line in the
project batch
file defines cell
libraries. See
page 45 to learn
more about
editable vs.
protected cell
libraries.

LIBS=ROOT (or just ROOT) will generate data for nested components only for cells contained in the same directory as the cell you are currently editing.

To include nested component data only for cells in directories marked as editable, use LIBS=EDITABLE. To generate nested component data for all cells nested in the current cell, regardless of the directories the cell files are stored in, use the default of LIBS=ALL.

Use extreme care when executing a command file generated by a **SHOW** command using **LIBS=(ROOT | EDITABLE | ALL)**. Executing such a file will modify cell files in the indicated libraries. Duplicate components will be added to each of those cells. Following the two steps below will ensure that all cells created by the SHOW file will be created in a new directory and will not corrupt existing data.

- Launch ICED™ using a new, empty working directory. - This ensures that no cells in the same directory as the original cell can be corrupted.
- Use a special ICED™ project batch file to launch ICED™ with no cell libraries defined. - This ensures that ICED™ cannot modify any cell libraries

Example: **SHOW ALL LIBS=ROOT ...**

This is a fragment of a SHOW command. Let us say that you are editing a cell named PROTO. A cell with the name MYCELL is nested inside PROTO and the cell file for MYCELL is stored in the same directory as PROTO. Two other cells are also nested in PROTO. Those two cells (ANDGATE and ORGATE) are contained in other directories. The data provided by the SHOW command will look something like this:

! Components in PROTO:

```
EDIT CELL MYCELL
ADD TEXT="MYCELL TEXT" LAYER=TEXT ID=1 SIZE=2.000 ...
... JUST=LB AT (0.5,-5.0)
EXIT ! CELL MYCELL

ADD CELL "ANDGATE" ID=1 AT (1.0,-8.5)
ADD CELL "ORGATE" ID=2 AT (1.5, 30.5)
ADD CELL "MYCELL" ID=3 AT (-49.5, 1.5)
ADD WIRE LAYER=M1 ID=4 TYPE=2 WIDTH=3.000 ...
... AT (13.5,-0.5) (13.5,-14.5) (32.5,-14.5)
```

The first line of the SHOW report is a comment displaying the name of the cell SHOW is reporting on. The next block of lines is about the cell MYCELL. This block of lines would modify the existing MYCELL cell if you executed the lines as ICED™ commands when MYCELL is in the working directory or in a cell library defined in the current session. Each component in MYCELL is listed (there is only one, a text component).

The next block of lines lists all of the components in PROTO. Note that the ADD CELL "MYCELL" command, which would add the cell MYCELL to PROTO at the necessary coordinates, is included. The other two cells (ANDGATE and ORGATE) do have ADD CELL lines listed since they are components in PROTO. However, since their files are not in the directory of the root cell, they do not have EDIT blocks of lines that list all components contained in the cells. The last line is about a wire component that is in the cell PROTO. Note that all of the data needed to create the wire is included in the line created by the SHOW command.

If you executed the SHOW file shown above as a command file while editing a cell in the same directory as PROTO, it would edit the cell MYCELL. A duplicate text component would be added at the same coordinates as the original text component. This modified version of MYCELL would be saved when you exit, overwriting the original version of MYCELL. However, if this file was executed while editing a cell in a new empty directory (and the cell library with the existing MYCELL was **not** defined as a cell library in the current session), a new copy of MYCELL would be created in the new directory and the old MYCELL would remain intact.

[WIRE_OUTLINES]

The WIRE_OUTLINES parameter will cause ICED™ to add an additional line to the report on each wire component. This line will be in comment format and will begin with the prefix "! OUTLINE". The coordinates of the vertices on the corners of the wire outline will be listed.

[AREA_ONLY | COMMANDS_ONLY]

The AREA_ONLY and COMMANDS_ONLY parameters are optional and mutually exclusive.

Using AREA_ONLY will result in a report with no component creation commands. Only the component area, perimeter length, wire length, and which layer the component is on will be reported for each selected component in comment format. No data is reported for cells, arrays, text, or line components. (The AREA_ONLY keyword is incompatible with the ALL keyword.)

The COMMANDS_ONLY keyword will cause the SHOW command to include only the component creation commands. The area comment lines will not be included. This is the mode used by default when the ALL keyword is used.

When the SELECT keyword is used, and you do not include either of the AREA_ONLY or COMMANDS_ONLY keywords, both forms of data will be generated by SHOW.

[IDS | NOIDS]

The IDS keyword will cause the SHOW command to include the unique id of each component in the ADD command lines. Using NOIDS will prevent the id from being included. The ID parameters are ignored by ICED™ if the ADD commands in the SHOW file are executed.

The unique id of a component reported by SHOW can be used to select it for deletion or modification.

Example: **UNSELECT ALL**
 SELECT ID 256
 DELETE

This example deletes exactly one component with the unique id 256 from the design. The id was obtained by using the SHOW ID command.

[**BY_NUMBER** | **BY_NAME**]

See the **LAYER** command to associate layer names with layer numbers.

This choice of keywords controls how layers are listed by the SHOW command. BY_NUMBER will use layer numbers rather than layer names when reporting which layer a component is on. BY_NAME (the default) will indicate layers by the name associated with the layer by the LAYER command.

[**MACROS=match_string** | **#match_string** | **USER_MACROS=match_string** | ...
... **SYSTEM_MACROS=match_string**]

Use of these parameters will cause ICED™ to list current macro definitions in the report created by the SHOW command. (Macros are strings which when used in a command are replaced with a value before the command executes.) These keywords are intended to be used without the SELECT or ALL keywords being used in the same command, but both types of reports can be combined.

The MACROS keyword will report on all macros that match the *match_string*. The *#match_string* syntax is simply shorthand for **MACROS= match_string**.

Example: **SHOW #***

The command above will list all macros currently defined.

Using USER_MACROS instead will restrict the report to macros created by the user. Using SYSTEM_MACROS will restrict the report to macros created by the ICED™ program. These system macros include:

CELL	the name of the cell currently being edited,
JOU	the name of the journal file currently in use,
RES.STEP	the resolution grid step size,
VIEW.BOX	the current view window corner coordinates
and many others	

The **KEY** command is used to create macros which are assigned to keyboard keys.

User-defined macros are reported in command format. Each macro which matches *match_string* will have a macro definition command in the report. These macro definition commands (GLOBAL or LOCAL commands) can be used to create equivalent macros in a future ICED™ session.

The system macros will be reported in \$-comment format. The lines generated in the SHOW report for system macros will be ignored by the command interpreter when executed in a command file.

The *match_string* parameter restricts the search to macros which are identical to *match_string*. You can use the wildcard characters * and ? when creating *match_string*. The asterisk (*) represents any number of characters or none. A question mark (?) matches any one character.

Example: **SHOW SYSTEM_MACROS=S***

All system macros that begin with the letter 'S' will be included in the SHOW report which will look something like the following:

!Name	Value	Description
\$ SHORT.CELL	MYCELL	First 8 characters CELL
\$ SHORT.ROOT	MYCELL	First 8 characters CELL.ROOT
\$ SNAP.ANGLE	45	Snap angle
\$ SNAP.OFFSET	(0.0, 0.0)	Snap offset
\$ SNAP.STEP	(0.5, 0.5)	Snap step size
\$ START.CMD	Q:\ICWIN\NEW.CMD	Name of startup.CMD file

The \$ at the start of each line will cause the line to echo in the journal file when executed as an ICED™ command, but there will be no effect to the system macro settings when these lines are executed. You can never change the value of any system macro directly.

Examples: **SHOW SYSTEM_MACROS=***
SHOW SYS=*

These two examples are equivalent. Either will list all system macros. The first line shows the full command syntax. The second is equivalent shorthand. When you use simply an asterisk (*) for *match_string*, it will match any macro name.

[LIST=*list_match_string*]

Use the LIST keyword to list the names of named lists created with the LIST command. The use of wildcards in *list_match_string* is exactly the same as that in *macro_string* for the MACRO keyword.

[LINE_LENGTH=*length*]

The LINE_LENGTH keyword is used only when you are using SHOW to create a file. The *length* parameter controls the maximum number of characters printed on one line of the file. The default is 80 characters per line. This is desirable in most cases. However, if you will be using the output from SHOW for input to another program, you may prefer that the lines not be broken up with continuation characters and hard returns. In this case, use the LINE_LENGTH parameter to set *length* to a very large number. Valid values for *length* are in the range 70:7040.

(FILE=(*file_name* | *) | APPEND)

Either a FILE parameter or the APPEND keyword is **required** in a SHOW command.

If you use the FILE=*file_name* parameter, it is used as the name of the file created. If no extension is provided, ICED™ will append an extension of .SHO to *file_name*. If *file_name* does include an extension, that extension will be used. If *file_name* includes a path, the file will be created in that directory. If no path is provided, the file will be created in the current directory.

The current directory is the working directory unless you specify a different working directory path in the *cell_name* on the ICED.EXE command line.

An asterisk (*) should be used in place of *file_name* when you want the data displayed on the screen. If the listing does not fit in your window, you will see a scroll bar on the right side of the window. Use your mouse on the scroll bar or use the arrow keys to scroll the listing. When you are done looking at the data, hit the <Enter> key or both mouse buttons to return you to the layout view.

If you do not provide the FILE parameter in the SHOW command, you will be prompted to enter it. Once you are prompted with the phrase "Enter output file name or <enter> for screen display:", you can type in the *file_name* parameter or simply press the <Enter> key to display the data on the screen.

Use the APPEND keyword instead of a FILE parameter when you want the information produced by the SHOW command to be appended to a file created by a previous SHOW command.

Example: **SHOW SYS =RES.* FILE=RESMACRO**
 SHOW SYS=SNAP.* APPEND

(Remember that the SYSTEM_MACROS keyword can be shortened to SYS.) The first command will store a report on all system macros beginning with the string "RES." in the file RESMACRO.SHO in the working directory. The second command will add a report on all system macros beginning with "SNAP." to the same file.

Advanced Uses of the SHOW Command

See the
Command File
Programmer's
Reference
Manual for
more
information on
using the
SHOW
command in
command files.

The SHOW command can be used to modify components. For example, you need to change the wire width of several components. First, select the components, then use a SHOW SELECT command to create a file called NEWWIDTH.CMD. Use any ASCII editor to edit the WIDTH field of the ADD commands in NEWWIDTH.CMD. After saving the file and returning to ICED™, delete the selected old wires and execute @NEWWIDTH to add the modified wires.

See the
command files
supplied with
ICED™ in the
Q:\ICWIN-
\AUXIL
directory for
examples of
using SHOW in
a command file.

The SHOW command can be very powerful when used in a command file. Using SHOW is an effective way of passing ICED™ data to an editor or to an application program for complex processing. (The ED.CMD command file, supplied with ICED™, is an example of passing data to an editor. ED.CMD is basically an automated version of the process outlined in the previous paragraph.) The ADD commands in the file generated by SHOW can be parsed by an application program and manipulated, then the modified file is executed to add the modified components to the design.

SNAP

Define a temporary grid for digitizing coordinate data.

```
SNAP [ STEP=x_step_size [, y_step_size] ] ...  
... [ OFFSET=offset_position ] ...  
... [ ANGLE=angle ]
```

The SNAP command is used to define the grid used when digitizing points with the mouse. The grid set with this command is related to the resolution grid set with the RESOLUTION command. The resolution grid is usually not altered once set. The SNAP command should be used instead if you want to temporarily digitize points on a grid more restrictive than the resolution grid. By default, the snap grid is identical to the resolution grid.

When the mouse is used to define points on the ICED™ screen, it will snap those points onto the grid set with the SNAP command. The mouse cannot be used to digitize a point that is not on the snap grid.

The CURSOR SNAP=ON command forces the cursor to jump between points on the snap grid.

For instance, when digitizing the vertices of a polygon with the mouse, the cursor may travel between points on the snap grid, but when the left mouse button is pressed, the point is digitized on the nearest point on the snap grid.

The snap grid affects only points digitized with the mouse. If you type in coordinate data yourself using a command like **ADD BOX AT (0.05, 0.05) (10.05, 12.05)**, the points will **not** be rounded to lie exactly on the snap grid. Points which are calculated by commands like ADD CIRCLE are also not forced to lie on the snap grid. Changing the snap grid will **not** modify components that have already been created.

In the rare case when the RESOLUTION command is used to change the resolution grid, the snap grid will be reset to have the same step as the new resolution grid, and the offset is reset to (0.0, 0.0)

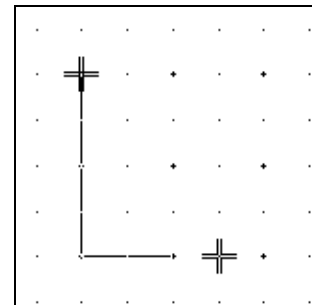


Figure 109: Points can only be digitized on the snap grid (crosses) even though the cursor is placed on the resolution grid (dots).

The snap grid is not shown on the screen. The GRID command controls the display grids shown on the screen.

[STEP=*x_step_size* [, *y_step_size*]]

If the SNAP ANGLE (see below) is set to 45°, *x_step_size* must equal *y_step_size*.

The STEP parameter determines the space between points on the snap grid. Unlike the resolution grid, the step of the snap grid can be different in the X and Y directions. Use a pair of real numbers for the STEP parameter to set the X and Y steps in user units. If a single number is used, the X and Y steps are assumed to be the same.

Example:

SNAP STEP=(5,10)

This example sets a snap grid only allowing points to be digitized where the X coordinates are multiples of 5 user units. The only allowable Y coordinates for digitized points will be multiples of 10 user units. Some points which could be digitized using this grid are (5,10), (0,10), (10,-20), and (-100,-100). The point at coordinates (5,5) could not be digitized with the mouse since Y=5 is not on the snap grid.

Different steps in the X and Y directions are useful if you want to create horizontal wires with one pitch and vertical wires with a different pitch. (Pitch is defined as the distance between center lines of wires)

Both STEP values used by SNAP will be rounded to be positive integer multiples of the *step_size* set by the RESOLUTION command. This insures that points digitized onto the snap grid also lie on the resolution grid.

Example:

SNAP STEP=0

Since ICED™ rounds each SNAP step size to the nearest positive multiple of the resolution step, this command will reset both SNAP step sizes to be the same as the RESOLUTION step size.

[OFFSET=*offset_position*]

When defining the snap grid, you can define an offset position that will shift the origin of the grid by multiples of the resolution grid step size. *offset_position* must be a coordinate pair of real numbers in user units. Each offset coordinate must be an integer multiple of the resolution grid step size. This insures that points digitized on the offset snap grid also lie on the resolution grid.

Example: **SNAP STEP=5 OFFSET=1,2**

This snap grid will have an origin of (1,2) and valid grid points will be located at intervals of 5 units in any direction. Valid grid points would be located at:

$$\begin{aligned} x &= 5n + 1, \\ y &= 5m + 2, \text{ where } n \text{ and } m \text{ are any integers} \end{aligned}$$

Using this snap grid would allow the point (1,2) to be digitized using the mouse. The coordinates (1,7), (-4,-8), and (6,7) could also be digitized on this grid. The points (5,5) and (0,0) could not be digitized because they do not lie on this grid.

[ANGLE=*snap_angle*]

The snap angle does not affect shapes created with typed coordinates or shapes that have already been created.

The *snap_angle* parameter controls the allowable angles between successive points when using the mouse to digitize a wire, polygon, or line. The only valid values for *snap_angle* are 0, 45, and 90. A *snap_angle* of 0 indicates to ICED™ that all angles are allowed. If ANGLE=45, angles between successive points will be constrained to multiples of 45°. If ANGLE=90, angles between successive points will be constrained to multiples of 90°. In this case, all geometry created with digitized coordinate data must have sides that are horizontal or vertical, all corners must be 90°.

The *x_step_size* and *y_step_size* of the snap grid must be equal if a *snap_angle* of 45° is used.

See the tutorial on "Creating Components at Arbitrary Angles" in the Classroom Tutorials Manual to learn more about using SNAP ANGLE=0.

Keyword Order

The order of the keywords is unimportant when using the SNAP command. If keywords are absent, their values will not change.

Example: **SNAP**

This form of the SNAP command reports the current settings of the snap and resolution grids. No changes are made to either grid.

SPACER

Modify spacing cursor for adding wires or polygons.

```
SPACER [ ON | OFF ] ...
... [ SPACE=spacing_distance ] ...
... [ TRACK_LAYERS=( ON | OFF ) ] ...
... [ LAYER=(layer_id | * ) ] ...
... [ STYLE=style_number ]
```

or

```
SPACER spacing_distance
```

or

```
SPACER
```

Use the **LAYER** command to specify a default *spacing_distance* for each layer.

The SPACER command controls the appearance of the cursor when you are adding new wires, boxes, or polygons. The cursor can include various guides to aid you in placing new components at least a minimum distance away from existing components.

This command will not prevent you from violating any minimum spacing requirements. It merely controls a visual aid that helps you avoid violating spacing rules.

When the spacer mode is ON and you are digitizing points during an ADD command for a new wire, box, or polygon, the cursor will have guide marks *spacing_distance* from the edge of the new component. If you align these guide marks with the edges of an existing component, then the two components will be exactly *spacing_distance* apart. The appearance of the guide marks is controlled by *style_number*.

The TRACK_LAYERS and LAYER keywords are used to set the *spacing_distance* value using a default specified with the LAYER command.

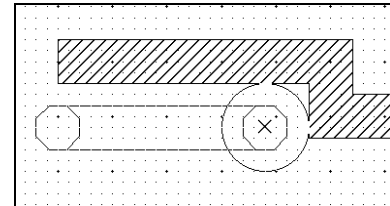


Figure 110: Using the spacer cursor to add a wire

[ON | OFF]

These keywords are used to turn the spacer mode on or off.

See page 327 for a few special situations where the spacer cursor will not be drawn.

You do not need to turn the spacer mode off when you are not adding components; the spacer cursor is only drawn when digitizing positions for certain types of ADD commands. However, the spacer cursor is somewhat complex and you may wish to turn off its display when not working in areas where exact spacing is important, or when you are working at high magnifications and can see the spacing between components easily.

[SPACE=*spacing_distance*]

When you use the SPACE keyword to set spacing, the *spacing_distance* parameter must be a non-negative real number of user units. Setting *spacing_distance* to 0 will prevent the display of the spacing cursor.

Example: **SPACER ON SPACE=1.5**

You can use a shorter syntax for the **SPACER** command when all you need to do is set the *spacing_distance*. See page 327.

This command will turn the spacing cursor on. The *spacing_distance* is set to 1.5 user units. When you execute an ADD command for a new wire, box, or polygon, the spacing cursor will make it easier to place the new component so that the edges are at least 1.5 user units away from other components.

Use the TRACK_LAYERS or LAYER keywords instead of the SPACE keyword when you want to use the *spacing_distance* already assigned to a layer by the LAYER command.

[TRACK_LAYERS=(ON | OFF)]

For most commands, ENABLED or YES can be used in place of ON. DISABLED or NO are equivalent to OFF.

Use this keyword to control the tracking mode of the SPACER command. The tracking mode controls whether the *spacing_distance* should be changed automatically every time the default layer changes. (The default layer is the layer used by the ADD command when you do not specify a layer in the ADD command. The default layer is changed by the USE or LAYER commands.)

When the tracking mode is ON and you change the default layer, the SPACER *spacing_distance* is automatically changed to the default *spacing_distance* for the new default layer. You will not have to execute any SPACER command to change the *spacing_distance*.

If tracking is on, you can still change the *spacing_distance* manually with a SPACER command. However, any value for *spacing_distance* that you set manually is overridden automatically when you change the default layer.

Example:

You must actually change the default layer to a different layer for the tracking mode to affect the cursor spacing. If the default layer was already M1 before this example was executed, the *spacing_distance* for the first ADD WIRE command would remain 2.0.

```
LAYER M1 SPACE=1.5
LAYER M2 SPACE=1.5
SPACER ON SPACE=2 TRACK_LAYERS=ON ! spacing_distance = 2.0
LAYER M1 ! default layer now M1, spacing_distance = 1.5
ADD WIRE
SPACER 2.5 ! spacing_distance = 2.5
ADD WIRE
LAYER M2 ! default layer now M2, spacing_distance = 1.5
ADD WIRE
```

The first and third ADD WIRE commands will use spacer cursor with a *spacing_distance* of 1.5 user units. The second ADD WIRE command will use a spacer cursor with a *spacing_distance* of 2.5 user units.

Notes:

The mode set with TRACK_LAYERS has no effect on the cursor spacing distance when you supply the layer for a new component in an ADD command. You must change the default layer to change the cursor spacing distance.

Unless you customize your startup command file, all layers have a default SPACE parameter of 0. You should add appropriate SPACE parameters to each wiring layer definition in your startup command file if you plan to make use of the TRACK_LAYERS feature.

[**LAYER**=(*layer_id* | *)]

The LAYER keyword is used to set *spacing_distance* to the default spacing of the layer identified by the *layer_id*. When you use * instead of a layer name or number, *spacing_distance* is set to the default spacing for the current default layer.

Example:

```
LAYER M1 SPACE=1.5  
SPACER ON TRACK_LAYERS=ON  
:  
LAYER M1           ! default layer now M1, spacing_distance = 1.5  
ADD WIRE  
SPACER 2.5         ! spacing_distance = 2.5  
ADD WIRE  
SPACER LAY=*       ! spacing_distance = 1.5  
ADD WIRE
```

Let us assume that the first two commands are in your startup command file. The commands that follow are typed while you are editing a cell. You do not need to rely on your memory to set the minimum spacing for layer M1. The LAYER M1 command automatically sets the spacer cursor to the correct *spacing_distance* for the layer. The first SPACER command then temporarily sets *spacing_distance* to a more restrictive minimum space for a critical wire. When you need to digitize non-critical wires, the SPACER LAY=* command returns *spacing_distance* to the correct default for the M1 layer.

[**STYLE=***style_number*]

The *style_number* must set to an integer in the range [1:4].

If you have never changed the *style_number*, the default style number 1 will be used. For boxes and polygons, style number 1 is a simple circle with a radius of *spacing_distance* around the standard cursor. See the table below for all available spacing cursor styles.

All styles include the standard wire cursor when you are adding a wire. In this case the radius of the spacing cursor circle is *spacing_distance* plus the half width of the wire. This cursor is sized correctly to place the **edge** of the wire at least *spacing_distance* from another shape. This idea works best with type 2 (extended-end) wires. When adding type 0 (flush-end) wires, the wire cursors for styles 3 and 4 are somewhat simpler than those shown below since the ends of type 0 wires do not extend to the edge of the wire cursor.






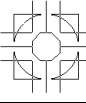

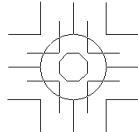
SPACER <i>style_number</i>	Cursor for boxes and polygons	Cursor for wires
1		
2		
3		
4		

Figure 111: Spacing cursor styles

Shorthand SPACER Syntax

The shorter versions of the SPACER syntax are used to save keystrokes.

When you need to set only *spacing_distance*, no keywords are required. When you supply a number as the only parameter, the *spacing_distance* is set to the number. This shorthand of the command also assumes that you want to use the spacer cursor so it sets the spacer mode to ON.

Example: **SPACER ON SPACE=1.5**
 SPACER 1.5

These two commands are exactly equivalent. The *spacing_distance* is set to 1.5 user units.

Example: **SPACER**

When you type the SPACER command with no parameters, it toggles the spacer mode from ON to OFF, or from OFF to ON. It also reports the other SPACER settings at the bottom of the window.

Since you may want to toggle the SPACER mode fairly often while thinking about other things, you may find it useful to make it even easier to toggle the SPACER mode. This can be accomplished by assigning the SPACER command to a key.

Example: **KEY F8 = SPACER**

The **KEY** command assigns commands to function keys.

Once this command is executed, pressing the F8 function key will toggle the SPACER mode. You can add this command to your startup command file so this function key assignment is made in every cell you edit.

Cases where the Spacing Cursor is NOT Drawn even when the Mode is ON

If *spacing_distance* is set to 0 the spacing cursor is not drawn. Remember that this may be the case if LAYER_TRACKING is ON and you change the default layer to a layer that has a default SPACE parameter of 0.

If the *spacing_distance* translates into fewer than 3 screen pixels, the spacing cursor is not drawn.

If the current SNAP setting (in either the x or y directions) translates into fewer than 1.5 screen pixels, only the simple circle spacing cursor will be displayed. This is intended to prevent the illusion that you can digitize coordinates more accurately than the screen resolution allows.

If you have a very large display screen, there are technical reasons that prohibit the display of **very** large (greater than 8000 screen pixels) spacing cursors.

SPAWN

Launch console application without a wait

SPAWN [-]

or

SPAWN [^][-]dos_cmd_string

The SPAWN command is used either to execute other applications or to open a separate window running the MS-DOS command interpreter. It is very similar to the DOS command that also launches an application, however the DOS command halts the editor until the application is complete. The SPAWN command allows you to continue to work in the editor while the application is running.

Example:

SPAWN

When you use the SPAWN keyword with no command string, a new window is created running the DOS command interpreter. You can then enter any number of valid console commands (e.g. the DIR MS-DOS command, or the MkPlot utility). The layout editor is still running and you can continue to execute ICED™ commands without closing the new window. When you are finished, type "EXIT" at the console prompt and the window will close.

Any changes to the current directory, disk, or environment variables in the console window will not affect settings in the layout editor. To see more details on the environment (e.g. the ICED_PATH and PATH environment variables) set up for the new console window, refer to the DOS command on page 184.

You can type a valid console application command string in the SPAWN command and it will launch that application in the new window. If the command contains characters that may confuse the command parser (e.g. use of '&', '+', or quotes), surround the *dos_cmd_string* (including any optional prefixes) with quotes.

The four valid quote characters are ", ', ~, and `.

The *dos_cmd_string* can refer to an application such as a text editor. Suppose you are working on a cell named MYCELL and your favorite text editor is NOTEPAD.EXE. If you want to see the last few ICED™ commands you executed you can type the following command:

Example:

SPAWN NOTEPAD MYCELL.JOU

The menu option 1:FILE->edit.JOU executes this SPAWN NOTEPAD ... command.

A command similar to the one above will open a new console window with the contents of the current journal file displayed. You can scroll to the bottom of the file to see the last several commands. (Be careful to not save changes to the journal file. Changing the journal file while ICED™ is still running is not a good idea.)

You can re-execute a command (or command file) by pressing the arrow keys. See the **ARROW** command.

You can also copy the last few commands into a new command file. Save the new file and leave the editor open while you execute the file with the *@file_name* command in the layout editor. You can then make a few changes to the file in the text editor, save the file again and re-execute it in the layout editor without opening or closing any windows. When you complete debugging the new command file, exit the editor and the console window will close automatically.

The Optional ^ Prefix

When you want to launch a single console application command by typing SPAWN *dos_cmd_string*, you can precede the string with a '^' character to minimize the console window while the command executes.

Example:

SPAWN ^COPY MYCELL.SF A:

The command above will copy the stream file MYCELL.SF to your A drive. This command may take a few moments to complete, but the layout editor will not be halted while it completes. No visible window will be created.

The Optional – Prefix

Use this prefix when you want to launch a Windows application that does not require the creation of a console window or the ICED™ environment variables.

In Windows 95 and Windows 98, environment variables must be defined with a batch file before a console window is opened. In these operating systems a batch file with the name W95\$ENV.BAT will be created in the Q:\ICWIN\²⁶TMP directory. ICED™ will execute this file by default when the '-' prefix is not used to create the appropriate environment prior to opening the console window or executing *dos_cmd_string*.

See the table on page 43 to learn more about the ICED™ environment variables.

The '-' prefix also prevents the creation of a console window and the execution of the MS-DOS command interpreter. If the previous example had used '-' instead of '^' as a prefix, the COPY command would fail since the Windows operating system needs the command interpreter to process an MS-DOS command like COPY.

Do not use the '-' prefix if you are executing ICED™ utilities or other programs that require either the command interpreter or the ICED™ environment variables.

Example: **SPAWN -CALC**

CALC.EXE is a GUI (Graphic User Interface) program that simulates a calculator. This standard Windows utility does not require the creation of a console window or the execution of the command interpreter. So the '-' prevents the creation of an extra unnecessary window.

The command above will open a new window running the CALC.EXE application. This allows you to perform calculations. The layout editor will not be halted while this window is open. You can switch back and forth between windows as required. Closing the layout editor will not close the calculator window, you must close it explicitly.

²⁶ Remember that Q:\ICWIN represents the drive and path where you have installed ICED™.

You can assign a command like this to a keyboard key to open the calculator from the layout editor with a single keystroke. Refer to the **KEY** command.

STREAM *Output design as a Stream (Calma-GDSII compatible) format file.*

```
STREAM ( MAP_LAYERS | ARCHIVE ) ...  
... [ FULL | ROOT | NOLIBS ] ...  
... [ SCALE=scale ] ...  
... [ SAMPLE=sf_file_name ] ...  
... [ FORCE_FLUSH | WIRE_TYPE=FLUSH | WIRE_TYPE=PRESERVE ] ...  
... [ FONT=font_name ] [ HEIGHT=font_height ] ...  
... [ CASE=( UPPER | LOWER ) ]
```

See page 337 to see where stream files for protected cells will be stored.

The STREAM command is used to output ICED™ design data as a Stream format (Calma-GDSII compatible) file. The Stream output file is usually placed in the same directory as the cell you are editing. The file name will be the cell name with a .SF extension.

(MAP_LAYERS | ARCHIVE)

Exactly one of the two keywords, MAP_LAYERS or ARCHIVE, must be used in the STREAM command.

The MAP_LAYERS keyword will cause ICED™ to use the stream layer numbers, data types, and text types assigned by the LAYER command when the Stream file is created. This is the option to use when creating mask data.

Use the **TEMPLATE** command to display the current Stream layer parameters.

The STREAM keyword of the LAYER command is used to assign Stream layer numbers to ICED™ layers. Data types and text types can also be assigned to layers using this keyword. Data types and text types provide tags that can be used by other programs, such as design rule checkers.

Read **A Special Note on CIF and STREAM** at the end of the **LAYER** command description.

When the MAP_LAYERS keyword is used, the STREAM command will only output components on layers that have been assigned non-negative Stream layer numbers. ICED™ deliberately forgets the current Stream layer numbers when you exit ICED™. If you do not reassign them before executing the STREAM command you will get the error message similar to the following:

"No layers assigned Stream layer numbers."

See page 25 for more information on the startup command file.

This behavior is intended to prevent accidents in which Stream files (and mask sets) are created using an obsolete ICED-Stream layer correspondence. A convenient way to assign Stream layer numbers is to include them in your startup command file. You can execute the startup command file by typing the @* command.

Using the ARCHIVE keyword will cause the assignments made by the LAYER command to be ignored. When the ARCHIVE keyword is used, ICED™ layer numbers will be used for the stream layer numbers, and all components will be assigned 0 for data types or text types. Components on all layers will be included. This form of stream file is useful to archive and transport design data.

[FULL | ROOT | NOLIBS]

These keywords determine which subcells are output to the Stream file. The default is FULL. The FULL keyword instructs ICED™ to include data for the cell you are editing and all of its subcells in the Stream file. The ROOT keyword instructs ICED™ to include data for the cell you are editing but **not** to include data for any of its subcells. Finally, the NOLIBS keyword, instructs ICED™ to include data for the cell you are editing and all subcells in the same directory but **not** to include data for subcells in other directories.

[SCALE=*scale*]

When you supply a coordinate in ICED™ or when ICED™ prints a coordinate on the screen, the value is given in user units. The *scale* value is the size of the user unit in microns. If you do not supply the SCALE parameter, it defaults to one micron per user unit.

[SAMPLE=*file_name*]

See the **NDIV** command line parameter for more details on units.

Both ICED™ and Stream use an integer database (i.e. all coordinate values and measurements are stored as integer multiples of a database unit). In ICED™, one user unit is divided by the NDIV parameter to get one database unit. All values are stored as multiples of this database unit. In Stream files, there is an explicitly defined Stream database unit and a Stream user unit. By default, ICED™ uses 0.001 microns for the Stream database unit and 1.0 micron for the Stream user unit. You can use the SAMPLE option to override these defaults.

The SAMPLE parameter does **not** affect the physical size of the chip geometries, just the units used to express them.

The size of the Stream database unit and Stream user unit are recorded in the Stream file as floating point numbers. Unfortunately, floating point arithmetic is subject to a variety of round off errors. Thus, if one creates pieces of a cell library on two different systems, there is a good chance that the unit sizes on the two systems will be very slightly different. The differences should be small enough so that they are not physically significant. Nevertheless, this may still lead to problems when you try to combine work from the two systems.

ICED™ tries to bypass this problem by copying the UNITS record from an existing Stream file. This procedure guarantees that the units in the two files are identical.

Example:

STREAM MAP SCALE=25.4 SAMPLE=Q:\ICWIN\AUXIL\MILS.SF ...

This example is a fragment of a STREAM command used to output a database in mils (1 mil = 0.001 inch = 25.4 microns). The SCALE parameter tells ICED™ that one ICED™ user unit represents 1 mil. This parameter fixes the physical size of the design. The SAMPLE keyword is used to set the size of the units used in the Stream output file. This does **not** affect the size of the design, only the units used to describe it.

The file Q:\ICWIN\AUXIL\MILS.SF²⁷, created on a Calma GDSII system, is supplied with the ICED™ distribution. It uses 0.001 mils for the Stream database unit and 1.0 mil for the Stream user unit.

²⁷ Q:\ICWIN\ represents the drive and path where you have installed the ICED™ program files.

Notice that the *file_name* parameter includes a path in the above example. If no path is given, ICED™ looks for the file in the working directory. The .SF file extension is optional. If a different extension is given, ICED™ ignores it and replaces it with .SF.

Note also that the MAP_LAYERS keyword can be abbreviated to MAP. Most ICED™ keywords can be abbreviated to the first several letters of the keyword as long as the abbreviation is uniquely recognizable and cannot be confused with another valid keyword in the command.

[FORCE_FLUSH | WIRE_TYPE=FLUSH | WIRE_TYPE=PRESERVE]

By default, ICED™ outputs extended end wires as extended end paths. Unfortunately, some systems do not handle extended end wires properly. If the FORCE_FLUSH keyword appears in the STREAM command, ICED™ will convert extended end wires to equivalent flush end wires. This option does **not** affect the geometry of the wires.

The WIRE_TYPE=FLUSH and FORCE_FLUSH options are equivalent. The WIRE_TYPE=PRESERVE option (the default) will allow ICED™ to use extended end paths in the output Stream file.

[FONT=*font_name*] [HEIGHT=*font_height*]

These parameters tell ICED™ the name and height (in microns) of the text font used on the GDSII system. The defaults used, if the HEIGHT or FONT parameters are not provided in the STREAM command, are 1 micron and "GDSII:CALMAFONT.TX".

[CASE=(UPPER | LOWER)]

Refer to the **SFMap** utility to learn about lower or mixed case structure names in stream files.

The CASE parameter controls whether cell names in the Stream file will be in upper or lower case text. The default is to use upper case. If you will be using the Stream file on a UNIX system, you may want to use the LOWER keyword. Otherwise, simply use the default by not including the CASE keyword in the STREAM command.

Location of the Output File

In previous versions of ICED™, when you executed the STREAM command during a nested edit (begun with the EDIT, P_EDIT, or T_EDIT commands), the output file was always written to the directory in which the subcell was stored. This was true even if this directory was a read-only or copy edit library.

See page 45 to learn about read-only or copy-edit libraries.

This version of ICED™ will create the output file in the subcell's directory only when that directory is defined as a direct edit library. If the subcell's directory is defined as a read-only or copy edit library, the output file is created in the working directory.

In older versions, the MAX_LAYER=*layer_no* parameter had to be used to allow layer numbers in the range 64:255 to be created in the stream file. Now all layers are automatically included.

SWAP

Swap layers of components or replace cells.

SWAP LAYERS *layer0_id* **AND** *layer1_id*
SWAP CELLS *cell0_name* **AND** *cell1_name*

SWAP LAYERS *layer0_id* **AND** *layer1_id*

The SWAP LAYERS command changes the layer of all fully selected components on *layer0_id* to *layer1_id* and all fully selected components on *layer1_id* to *layer0_id*. Partially selected components are unaffected. When only components on one layer are selected, the SWAP LAYERS command has the effect of changing the layer of those components without affecting any components on the other layer.

The layer ids can be layer names or layer numbers. Layer lists are not supported by the SWAP command. The AND keyword is required.

Example: **SWAP LAYERS NDIF AND PDIF**

In this example, all components on NDIF which were fully selected will be changed to layer PDIF. All fully selected PDIF components will be changed to layer NDIF. Partially selected and unselected components on those layers will not be affected.

Example: **UNSELECT ALL**
SELECT LAYER MIX ALL
SWAP LAYERS MIX AND M1
UNSELECT ALL

The PROTECT and BLANK commands prevent components from being selectable.

This series of commands will have the effect of changing the layer of all unblanked and unprotected components in the current cell on layer MIX to layer M1. No components previously on layer M1 will be affected.

SWAP CELLS *cell0_name* AND *cell1_name*

The SWAP CELLS command replaces all selected cells and arrays of cells with the name *cell0_name* with cell *cell1_name*, and all selected *cell1_name* cells and arrays of *cell1_name* cells with *cell0_name*. Unselected cells and selected cells or arrays with other cell names are unaffected.

Example: SWAP CELLS NPN AND PNP

In this example, all cells with the name NPN that were selected prior to the command will be replaced with cells with the name PNP. All selected PNP cells will be replaced with NPN cells.

Interactions between the SWAP and SELECT Commands

See the **NEAR** command to set the size of the NEAR box.

If you issue a SWAP LAYERS command and no components are selected, ICED™ will automatically generate an embedded SELECT LAYERS *layer0_id+layer1_id* NEAR command. Clicking the left mouse button will select all components on the two indicated layers within the near box whose center is the cursor.

If you issue a SWAP CELLS command and no components are selected, ICED™ will automatically generate an embedded SELECT CELL * AT command.

If you replace a cell with a larger cell, in some rare cases the new cell may extend beyond the maximum boundary the design area. If this happens the SWAP command will fail. You can use the SELECT FAIL command to select the components that caused the failure and unselect all other components. The SELECT POP command can be used to retrieve your original selections.

T_EDIT

Edit another cell with a transformed coordinate system.

T_EDIT [NOW] [LOCAL_COPY=(TRUE | FALSE)]...
 ... [VIEW_ONLY=(TRUE | FALSE)] (SELECT | NEAR *position*)

The T_EDIT (Transformed EDIT) command allows you to suspend work on the cell you are currently editing and edit another cell. (We will call these cells the parent cell and the child cell. We will call the cell you were editing when you first launched ICED™ the root cell.)

See the chart in the **EDIT** command for a comparison of edit commands.

T_EDIT is different from EDIT in that the coordinate system of the parent cell is used instead of the coordinate system of the child cell. The origin of the child cell will not be at the coordinate (0,0) while you are editing it. If the specific instance of the child cell you are editing was placed at the coordinate (100.5, 65.5) in the parent cell, then that will be the coordinate of the child cell's origin while you are editing it. If the child cell is rotated in the parent cell, that will be the orientation of the child cell as you edit it.

You cannot edit the child cell by name. Note that the EDIT command's CELL keyword is not a valid keyword of the T_EDIT command. This is because a specific instance of the child cell, already placed in the parent cell, is required for the T_EDIT command.

Each cell file contains two databases. The geometric database contains a description of each component; its type, position, select status, etc. The environment database contains information about the cell as a whole; layer names, layer colors, grid settings, etc. When you use the T_EDIT command to enter a cell, ICED™ loads only the cell geometry database. The child cell's environment database is ignored. Thus, if layer 1 is red in the parent cell and blue in the child cell, it will appear red while you use T_EDIT to edit the child cell. If you save the child cell, its environment database is replaced with the of the root cell.

See the
P_EDIT
command.

Unlike the P_EDIT (edit in **p**lace) command, T_EDIT will display only the cell you are editing. (P_EDIT leaves the parent cell displayed on the screen, but does not allow you to modify it while you edit the child cell.)

When you are done editing the child cell, you can use the QUIT, EXIT, or LEAVE commands to return to the parent cell. If you QUIT the child cell, the changes you made will be lost. If you use EXIT or LEAVE, ICED™ will save your changes in RAM. (LEAVE will save the cell only if the geometry has been modified.) The modified cell file will be saved to disk when the editor terminates even if you eventually QUIT the root cell.

See **Journaling and Data Recovery** for details.

ICED™ does not update your cell files until you terminate the layout editor. Thus, if your system crashes or you execute the JOURNAL command, the cell file will not have been saved. However, your work is not lost. ICED™'s automatic journaling feature will allow you to recover the changes.

See page 45 to learn more about protected cell libraries.

If the child cell is in a different cell library than the root cell, you may be restricted from saving changes made to it. The T_EDIT command will prompt you with a warning message if the cell is in a protected library.

[LOCAL_COPY=(TRUE | FALSE)] [VIEW_ONLY=(TRUE | FALSE)]

These parameters are used mainly in command files. If you prefer to avoid having ICED™ display a prompt if the cell you wish to edit is in a protected library, you can use the LOCAL_COPY or VIEW_ONLY keywords. For more details see the explanation in the EDIT command description.

When the LOCAL_COPY or VIEW_ONLY keywords are used in a T_EDIT command, they should precede the SELECT or NEAR keyword.

SELECT

One of the keywords SELECT or NEAR must be used with the T_EDIT command.

Remember that the name of the cell you are currently editing is reported in the upper left corner of the ICED™ window at the end of the list of open cells.

The T_EDIT SELECT command allows you to edit a selected cell. Only cells can be selected when you issue this command. If there is more than one cell selected when the T_EDIT SELECT command is executed, ICED™ will assume you want to edit the selected cell with smallest area. If no cells are selected when the T_EDIT SELECT command executes, an embedded SELECT CELL * AT command is generated to allow you to select the cell.

T_EDIT SELECT can be used repeatedly to traverse the hierarchy of nested cells. That is, you can use T_EDIT SELECT to enter a cell placed in the parent cell, then use T_EDIT SELECT again to enter a cell placed in that cell. You can traverse down six levels using this method. Since the cell will not shift in location, scale, or orientation on the screen (as it would with the EDIT SELECT command), this process is fast and easy.

NEAR *position*

Using the NEAR keyword allows you to traverse several levels of cell hierarchy at a time. If *position* is the coordinate pair that identifies a point near **the boundary of a visible component** in a deeply nested cell, T_EDIT NEAR *position* will allow you to edit the deeply nested cell which contains the component.

It is more common to use the mouse to identify *position*. If it is not supplied with the T_EDIT NEAR command, you can select the deeply nested cell to edit by merely putting the cursor over an edge of a visible component in that cell and clicking the left mouse button.

The NOW Keyword and the ENTER.SUBCELL Macro

The following new feature is useful primarily if you use specialized command files and/or macros. This feature is described in more detail in the description of the EDIT command.

The ENTER-.SUBCELL macro is also executed by the EDIT and P_EDIT commands.

If an ENTER.SUBCELL macro is defined when a T_EDIT command is executed, the string stored in the macro is executed as a command string. If the macro does not exist, the T_EDIT command continues without any errors. Add the NOW keyword to the T_EDIT command to avoid executing the macro.

Example:

T_EDIT SELECT

When the command above is used to edit a cell, and the macro ENTER.SUBCELL exists, then the command string stored in the macro is executed as soon as the cell is opened. If the command string includes a @cmd_file command, all of the commands in the file are executed.

Example:

T_EDIT NOW SELECT

When the NOW keyword is added to the T_EDIT command, the ENTER.SUBCELL macro is ignored and the T_EDIT proceeds without executing any additional commands.

TEMPLATE

Display or save template parameters.

TEMPLATE (* | *file_name*)

See the
@file_name
command to
learn about
command files.

The TEMPLATE command is used to display or save in a file the current values of the template parameters. The template parameters are the settings of most of the user-definable parameters in the environment database of the current ICED™ session. Everything from key assignments to color, grid, and layer settings are listed by the TEMPLATE command.

The current
directory is the
working dir-
ectory unless
you specify a
different wor-
king directory
path in the
cell_name field
on the ICED-
.EXE command
line.

If you specify *file_name*, the template parameters will be saved in the file *file_name* as a series of commands. You can later execute the file using the **@file_name** command. If *file_name* does not include an extension, ICED™ will add the extension .CMD to the file name. If *file_name* does not include a path, the file will be written to the current directory.

If you use an asterisk (*) instead of *file_name*, the parameters will only be displayed on the screen. If the listing does not fit in your window, you will see a scroll bar on the right side of the window. Use your mouse on the scroll bar or use the arrow keys to scroll the listing. When you are done looking at the data, hit the <Enter> key or both mouse buttons to return you to the layout view.

Example:

TEMPLATE *

This example displays current values of the template parameters on the screen. This can be very useful for quickly determining the current values of parameters like those assigned by the LAYER command. The version number of the program is reported at the top of the listing.

The current
directory is the
working dir-
ectory unless
you specify a
different wor-
king directory
path in the
cell_name on
the ICED.EXE
command line

Using the TEMPLATE Command to Create a Startup Command File

The TEMPLATE command is useful in creating startup command files. The sample startup file Q:\ICWIN²⁸\TECH\SAMPLES\NEW.CMD was created using the TEMPLATE command.

This commenting of commands did not happen in previous versions of ICED™.

When using the TEMPLATE command to create a startup command file, you should keep in mind that command line parameters like the MENU setting could be overridden by commands in the startup command file when it is executed. To prevent the created command file from overriding command line parameters, some of the settings stored by the TEMPLATE command (e.g. MENU and DISPLAY_CELL_DEPTH) are commented out with a '\$' in front of the line. Lines turned into comments this way are not executed by ICED™. If you want these settings to be overridden by the commands in the startup command file when it is executed, use a text editor to remove the '\$' from the beginning of the line.

Example:

TEMPLATE ST_TEMPLATE.CMD

This example writes the template parameters to the file ST_TEMPLATE.CMD in the working directory. When this command file is executed in another cell, it will create identical layer properties, grid settings, color definitions, etc. However the command file will not load the same menu as the original cell unless you remove the '\$' from the line with the MENU command at the beginning of the ST_TEMPLATE.CMD command file.

See page 25 to learn more about startup command files.

²⁸ Q:\ICWIN represents with the drive and path where you have installed the ICED™ program files.

TEXT *Set parameters that affect how text components are added and displayed.*

TEXT [JUSTIFICATION=*just_code*] ...
... [DISPLAY_ORIGINS =(ON | OFF)] ...
... [LOWER_CASE =(ENABLED | DISABLED)] ...
... [MULTI_LINE_TEXT =(ENABLED | DISABLED)] ...
... [ORIENTATIONS =(2 | 4 | 8)]

The TEXT command controls how ICED™ text components are created and displayed.

[JUSTIFICATION=*just_code*]

Refer to the **ADD TEXT** command for more details on the importance of *just_code* and several examples of its use.

The USE command can also be used to set the default text justification code.

This parameter sets the default text justification used by the ADD TEXT command. The value of *just_code* will determine which point of the text component will be used as the true location of the text component. If the text is labeling a component, the point identified by *just_code* must lie inside the component boundary. See Figure 112 for valid values of *just_code*. These values are equivalent to the justification codes in CALMA-GDSII files.

The DEFAULT_JUSTIFICATION or TEXT-_JUSTIFICATION keywords may be used as synonyms for this keyword.

<i>just_code</i>	Default text justification
LB or BL	Bottom Left
LC or CL	Center Left
LT or TL	Top Left
CB or BC	Bottom Center
CC	Center
CT or TC	Top Center
RB or BR	Bottom Right
RC or CR	Center Right
RT or TR	Top Right

Figure 112: Text justification codes

Example: **TEXT JUST=LC**

After this command is executed, new text components will have the origin of the component fixed to the Left Center of the text components unless the justification is overridden in the ADD TEXT command. This point is marked with a diamond when the component is selected.

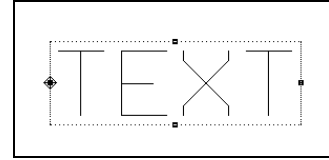


Figure 113: This text component was added with LC justification.

[**DISPLAY_ORIGINS** =(ON | OFF)]

For most commands, ENABLED or YES can be used in place of ON. DISABLED or NO are equivalent to OFF.

Previous versions of ICED™ displayed text origins only when a text component was selected. This version now optionally displays the origin of every text component whether or not it is selected.

The origin of a text component is determined by the value of the JUSTIFICATION=*just_code* parameter in place when the ADD TEXT command was executed. This origin can be any one of nine different places on the bounding box of the component. (See the previous page.)

For some uses (e.g. LVS labels) the origin of the text component is important. If the origin of the text component is not covered by the component it is meant to label, the text component will not be used as a label.

If you want to see these origins marked with a small cross in the display, enable this feature with the following command:

Example: **TEXT ORIGINS=ON**

Note that the command above demonstrates that the ORIGINS keyword can be used interchangeably with the longer (but more descriptive) keyword DISPLAY_ORIGINS.

This origin display can make center-justified text difficult to read, so if you prefer to not display the text origins with a small cross, leave the default of ORIGINS=OFF in place.

[LOWER_CASE=(ENABLED | DISABLED)]

The **USE** command can also be used to set default text justification and case.

This parameter enables or disables the use of lower case letters in text. If you set LOWER_CASE=DISABLED, the ADD TEXT command does not distinguish between upper case and lower case letters. Instead, it converts all letters to upper case. If you set LOWER_CASE=ENABLED, the ADD TEXT command distinguishes between upper case and lower case letters.

Many types of post-processing software rely on text to label objects. Some software distinguishes between upper case and lower case letters and some does not. To avoid confusion, we strongly recommend using LOWER_CASE=DISABLED.

[MULTI_LINE_TEXT=(ENABLED | DISABLED)]

Do not create multiline text components for conversion to maskable polygons. See the lesson on Maskable Text in the Classroom Tutorials Manual.

This parameter controls how text can be entered by the ADD TEXT command when you do not supply the string(s) in the command. In this case, the user is prompted for the string.

When the MULTI_LINE_TEXT mode is enabled, multiple lines of text can be entered at the “ENTER TEXT” prompt. You press <Enter> after the first line of text is entered, then type the next line. The <Enter> key must be pressed twice to complete the command. This is useful when adding large amounts of text to document your design, however pressing <Enter> twice for each text component is awkward when entering single strings.

If MULTI_LINE_TEXT is disabled, only one line of text can be entered. When <Enter> is pressed the command is completed.

[ORIENTATIONS=(2 | 4 | 8)]

See the **PLOT** command for details on orienting text in plots.

This parameter controls how text is displayed on the screen. It will affect the display of existing text components as well as new components. The geometry is not altered. The ORIENTATIONS parameter will not affect the way text is exported by the CIF or STREAM commands.

See page 144 for examples of the 8 possible display orientations.

ORIENTATIONS=8 means that text has 8 possible display orientations. All text will be displayed as it is oriented in the cell. Mirrored text will display as mirrored. Text rotated 180° will display upside-down and backwards.

ORIENTATIONS=4 will display mirrored text in an unmirrored state. Therefore, text will display in only 4 different orientations: horizontal right-side-up, horizontal upside-down and backwards, vertical forwards (i.e. the letters are read from down to up) or vertical backwards (read up to down).

ORIENTATIONS=2 will always display text as horizontal right-side-up or vertical forwards.

Keyword Order

The order of keywords is unimportant in the TEXT command. If keywords are absent, their values will remain unchanged.

Example:

TEXT

This use of the TEXT command will report the current settings.

UNDO

Reverse effects of the previous command.

UNDO

See **Journaling and Data Recovery** for other methods of recovering data.

The UNDO command can completely reverse the effects of the last command entered. Only the last single non-view command executed is affected by UNDO.

When UNDO is executed directly after the completion of a command file, since it cannot undo the effects of the entire command file, it will instead do nothing and warn you at the bottom of the window that it "Cannot UNDO".

Using UNDO repeatedly does not reverse the effects of the last several commands. Instead, UNDOing an UNDO command re-executes the original command.

Use the **VIEW LAST** command to undo a view command.

UNDO is intended to reverse the effects of a command that alters the design or the design environment. Commands which change the view window are unaffected by the UNDO command. Commands that change only the view window can come between an UNDO command and the command it undoes. This feature allows you to perform an edit command, change the view window to look closely at the results, then UNDO the edit command.

Example:

```
SELECT LAYER M1 ALL  
DELETE  
VIEW ALL  
UNDO
```

In this example, the effects of the DELETE command will be reversed by the UNDO, even though VIEW ALL was the previous command.

Some commands can not be reversed with UNDO. They include @file_name, EDIT, T_EDIT, P_EDIT, GROUP, PLOT, CIF, STREAM, VIEW LIMIT, EXIT, JOURNAL, LEAVE, and QUIT.

USE *Set default values for parameters used by the ADD command.*

```
USE [ LAYER=layer_id ] ...
... [ WIRE_TYPE=(0 | 2) ] ...
... [ ARC_TYPE=(0 | 2) ] ...
... [ N_SIDES=n_sides ] ...
... [ TEXT_JUSTIFICATION=just_code ] ...
... [ LOWER_CASE=(ENABLED | DISABLED) ]
```

Former name in some previous versions of ICED: **DEFAULTS**

The USE command is used to set the default values of parameters used by the ADD command. (Some of these parameters can also be set by other commands. This is noted below.) The meaning of these parameters is discussed in more detail in the section describing the ADD command.

[LAYER=*layer_id*]

The **LAYER** command can also be used to set the default layer.

This parameter informs the ADD command which layer to use as the default for adding components. The *layer-id* may be a layer name or layer number.

Any components (except cells and arrays) added to the drawing must be placed on a specific layer. If you do not specify the layer as part of the ADD command, they are placed on the default layer.

[WIRE_TYPE=0 | WIRE_TYPE=2]

Type 1 wires with rounded ends are not supported by ICED™.

This parameter sets the default wire type used by the ADD WIRE command. ICED™ supports wire types 0 and 2 which correspond to Calma path types 0 and 2. Type 0 wires have flush ends. Type 2 wires extend past the digitized points on the ends of the wire by half the wire width.

[ARC_TYPE=0 | ARC_TYPE=2]

This parameter sets the default wire type used by the ADD ARC command. ICED™ supports wire types 0 and 2 which correspond to Calma path types 0 and 2. (See WIRE_TYPE above for an explanation of wire types.)

[N_SIDES=*n_sides*]

This parameter sets the default value for N_SIDES used by the ADD RING, ADD CIRCLE, ADD SECTOR, and ADD ARC commands. Valid values for *n_sides* are integers in the range 3:198.

ICED™ uses many-sided POLYGONS to approximate CIRCLES and RINGS. It uses segmented WIRES to approximate ARCS. N_SIDES specifies the default number of sides used to approximate a full circle or a circular arc. If the ADD RING command is used with the default *n_sides* specified by the USE command, *n_sides* will be used as both the number of inner and the number of outer sides.

Example:

USE LAYER=POLY ARC_TYPE=0 N_SIDES=16

After this USE command has been executed, all ADD commands will default to layer POLY. New arcs will default to having flush ends. Also, new arcs will default to be segments of a circle with 16 sides. These defaults can be overridden by using the appropriate keywords on the ADD ARC command.

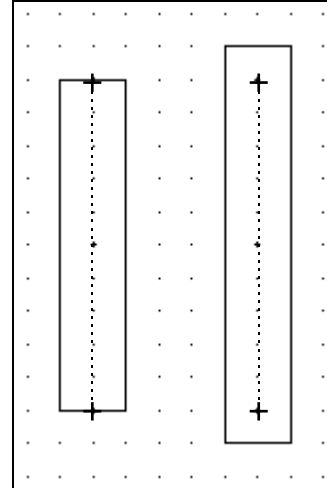


Figure 114: Both wires were created with the same distance between vertex points. The wire on the left is type 0. The right wire is type 2.

Many possible values of *n_sides* will be unusable by the ADD RING and ADD ARC commands. See those command descriptions for limitations on *n_sides* and for directions on overriding the default value specified by USE.

[TEXT_JUSTIFICATION=*just_code*]

See the table on page 346 for valid values of *just_code*.

This parameter sets the default text justification used by the ADD TEXT command. The JUSTIFICATION keyword may be used as a synonym for the TEXT_JUSTIFICATION keyword. The environment database setting specified by this keyword is the same as that set by the JUSTIFICATION keyword in the TEXT command.

[LOWER_CASE=(ENABLED | DISABLED)]

The **TEXT** command can also be used to set default text justification and case.

This parameter is used by the ADD TEXT command. It enables or disables the use of lower case letters in text. If you set LOWER_CASE=DISABLED, the ADD TEXT command does not distinguish between upper case and lower case letters. Instead, it converts all letters to upper case. If you set LOWER_CASE=ENABLED, the ADD TEXT command can create components with both upper case and lower case letters.

Many types of post-processing software rely on text to label objects. Some software distinguishes between upper case and lower case letters and some does not. To avoid confusion, we strongly recommend using LOWER_CASE=DISABLED.

Keyword Order

All the keywords in this command are optional. If any (or all) of them are omitted, the parameter values remain unchanged.

Example: **USE**

This form of the USE command (with no parameters) can be used to display the current USE default values on the screen.

VIEW

Set the view window.

```
VIEW [ ALL ] ...
    ... [ SELECT ] ...
    ... [ BOX pos1, pos2 ] ...
    ... [ CENTER pos ] ...
    ... [ MOVE disp ] ...
    ... [ LAST ] ...
    ... [ HOME ] ...
    ... [ SCALE view_scale ] ...
    ... [ IN factor ] ...
    ... [ OUT factor ] ...
    ... [ SAVE [LAST] window_no ]...
    ... [ WINDOW window_no ]
```

The **VIEW (ON | OFF)** command is used to set screen refresh during command files.

Use the **VIEW LIMIT** command to limit the amount of detail drawn in displaying large areas.

The VIEW command is used to set the view window. The view window determines the area of your drawing that will be displayed on your screen. (Two other commands begin with the keyword VIEW. See **VIEW (ON | OFF)** on page 360 and **VIEW LIMIT** on page 363.)

In general, the display area on your screen will be slightly larger than the area of the indicated view window. As a first approximation, ICED™ displays the smallest area of the drawing that both fills your display screen and includes the view window. The center of the display area will be near the center of the screen. ICED™ uses integer arithmetic to scale and display items in your drawing. Thus, there is a slight granularity in the scale factors it supports. This granularity is the reason for the approximation. The granularity is not normally apparent unless you are working at extremely high magnification.

Each of the keywords used to manipulate the view window is described below. They can be used in combination as well. See **Using Multiple Keywords** on page 358.

[ALL]

Use the **BLANK** command to turn off the display of individual components.

Use this command to display your entire drawing. The view window will be a box 1.05 times larger than the smallest box which surrounds all of the unblanked components in the drawing. If there are no unblanked components, the view window remains unchanged.

[SELECT]

The view window will be a box 1.05 times larger than the smallest box which surrounds all the selected components in the drawing. If there are no selected components the view window remains unchanged.

[BOX *pos1, pos2*]

The view window will be a box with corners given by the positions *pos1* and *pos2*. Both *pos1* and *pos2* should be coordinate pairs of real numbers. The order of the two positions is unimportant. If the positions are not included in the command, you must define them with the mouse.

[CENTER *pos*]

The view window will remain the same size as the current view window but its center is moved to *pos*. *pos* should be a coordinate pair of real numbers. If it is not provided in the command, you must define it with the mouse.

[MOVE *disp*]

See the AUTOPAN command to learn about automatic panning of the view window.

The view window will remain the same size as the current view window but *disp* will be subtracted from the center coordinates. *disp* should be a pair of real numbers. If *disp* is not provided, it can be defined with two mouse clicks. The displacement between the two cursor positions will be used as *disp*.

Example: **VIEW MOVE (25, 10)**

See the
ARROW
command to
learn about
using the arrow
keys to pan.

(25, 10) will be subtracted from the coordinates of the current view center. If the current view window is centered about the point (75, 40), the new view window will be centered about the point (50, 30).

[LAST]

Use this keyword to restore the previous view window. You can only back up one view window. If you use VIEW LAST twice in a row, you will restore the window that was used before the first VIEW LAST command was issued.

[HOME]

See the SAVE
keyword on
page 358 to
learn about
saving a
restoring a
specific view
window.

The VIEW HOME command can be used only from the nested view menu. It replaces the current view window with the view window in use at the start of the edit command.

Nested view commands can be used during any command which requires the mouse to digitize coordinates (except for VIEW commands). While you are digitizing coordinates for an edit command, pressing the <ESC> key, or the middle button of a three-button mouse, will bring up the nested view menu. You can alter the view window with any of the commands on the menu while the edit command is executing.

See the
AUTOPAN
command to
learn about
automatic
panning of the
view window.

During an edit command, you can enter any number of nested view commands, manual windows pans (using the arrow keys), or autopans (by moving the cursor off the screen). The VIEW HOME command will restore the window in place at the start of the edit command. VIEW HOME will have no effect once the edit command is complete. At that point, you can use VIEW LAST to restore the view window to what it was before the edit command was executed.

[SCALE *view_scale*]

Use this keyword to change the size of the view window so that the display scale in pixels per user unit is given by *view_scale*. *view_scale* must be a positive real number. The view center will remain unchanged.

Example: **VIEW SCALE 2.0**

If the ICED™ window is displayed maximized and you are using a VGA monitor with a display area 640 pixels wide, the view window is $640 - 48 = 592$ pixels wide. (The menu area is 48 pixels wide.) The view window will be $592 / 2.0 = 296$ user units wide.

[IN *factor*] and [OUT *factor*]

Use VIEW IN *factor* to make the view window smaller (zoom in) by *factor*, where *factor* is a positive real number. If *factor* is less than 1.00, it is replaced by $1 + \textit{factor}$. Thus, the commands VIEW IN 1.02 and VIEW IN .02 have the same meaning. The view center will remain unchanged.

Use VIEW OUT to make the view window larger (zoom out) by *factor*. If *factor* is less than 1.00, it is replaced by $1 + \textit{factor}$. Thus, the commands VIEW OUT 1.02 and VIEW OUT .02 have the same meaning. The view center will remain unchanged.

Example: **VIEW IN 5**
 VIEW OUT 5

The first command will zoom in by a factor of 5. If the view window was 200 user units wide, it will be 40 user units wide after the first command. The second command zooms out by the same factor. So after the second command, the view window will again be 200 user units wide.

[SAVE [LAST] *window_no*] and [WINDOW *window_no*]

The VIEW SAVE *window_no* command will save the current view window coordinates so they can be restored later. *window_no* must be in the range 1:9. The saved view windows are retrieved with the VIEW WINDOW command.

The optional LAST keyword saves the last view window instead of the current view window. This can be useful when you decide you might need to restore a view window after you have already changed it. Let us say you zoom in to look at a small area of your design, then decide you still need to zoom again. You want to be able return to the original unzoomed view window. You can use the VIEW SAVE LAST *window_no* command to save the original view window even though your current window has already changed. Now you can change the view window several times, then use VIEW WINDOW *window_no* to restore the original view window.

Example: **VIEW SAVE LAST 8**

is equivalent to the following 3 commands:

**VIEW LAST
VIEW SAVE 8
VIEW LAST**

Nine sets of view windows can be saved at a time. These view windows are saved with the cell file. Therefore, the VIEW WINDOW command can restore a saved view window in another ICED™ session at a later date.

If you use VIEW WINDOW with a window number which has never been saved with a VIEW SAVE, ICED™ will report an error.

Using Multiple Keywords

More than one view keyword can be used in the same command. ICED™ reads the keywords from left to right. If a BOX, CENTER, or MOVE keyword is used, it should be the last keyword in the command. The remaining keywords can appear in any order that makes sense.

Example: **VIEW ALL OUT 1.5**

In this example, the final view window will be the same as the one that would be generated by executing the two commands "VIEW ALL" and "VIEW OUT 1.5". Using one command will make the response time much faster since the display is not redrawn in between the two operations.

VIEW (ON | OFF) *Control display refresh during execution of command files.*

VIEW (ON | OFF)

See *@file_name*
for details on
executing
command files.

Normally, ICED™ updates the display after each command it executes. The display update usually takes most of the time required to execute the command. The VIEW ON and VIEW OFF commands allow you to enable or inhibit the display update during the execution of command files. The VIEW OFF mode, which inhibits display update during command files, results in an order of magnitude decrease in execution time. This is the default for new cells.

See the LOG
SCREEN
commands to
control the
update of the
status line
during
command files.

When the view mode is OFF, commands in a command file will not be displayed on the command line near the bottom of the window while they are being executed. If geometry is created or modified, this will not be reflected in the display until the command file is completed, or the VIEW mode is set to ON. (\$Comments are seen on the status line in either view mode.)

Commands in a command file that require the user to select or digitize components temporarily set the view mode set to ON. You no longer need to add VIEW ON commands to ensure that the view is updated correctly during these types of commands. However, a simple PAUSE command will not set the VIEW mode to on. If you want the display to reflect recent changes made by the command file, use a VIEW ON command before the PAUSE.

Example:

```

VIEW OFF
WHILE (%COUNT <=32000){
    :
    :
    VIEW ON
    $ %COUNT components added. Press both mouse buttons to abort.
    PAUSE
VIEW OFF
    :
    :
}

```

See the **Command File Programmer's Reference Manual** to learn about the **WHILE** command and using macros.

The command file fragment on the previous page consists of a while loop that may execute 32000 times. (The references to %COUNT are macro references that operate basically like variables in programming languages.) The block of lines in the middle allows the user to see the progress of the command file and abort it if it is not performing as expected. If VIEW commands were not included to temporarily turn the view mode on, the view window would not have the current geometry displayed while the command file is paused waiting for a response from the user. The VIEW OFF command is included to insure that screen updates do not slow the execution of the remainder of the command file.

When ICED™ executes a VIEW ON or VIEW OFF command outside of a command file, the default view mode for all command files executed in the current session is modified. When one of these commands is executed in a command file, it affects the view mode only during the current command file. In either case, this command does **not** affect the automatic refresh after every command entered from the keyboard or menus.

When ICED™ executes a command file, it begins execution with the current value of the view mode. This value remains in effect until it is changed by another VIEW ON or VIEW OFF command. When ICED™ exits a command file, it resets the view mode to what it was before entering the command file. Thus, a VIEW ON or VIEW OFF command in a command file affects only the view mode for the current command file and any command files it calls.

There are two other commands that begin with the keyword VIEW. The VIEW command (see previous command) is used to alter the view window. The VIEW

LIMIT command (see next command) is used to speed screen refreshes by controlling which components are drawn.

VIEW LIMIT

Speed screen refresh by limiting display detail.

VIEW LIMIT [ON | OFF] ...
... [SCALE=*view_scale*] ...
... [DEPTH=*depth*]
... [UNITS=*units*] ...
... [DOTS=*dots*] ...
... [SHOW_LAYERS=*layer_list*]

See the **VIEW** command for various ways of zooming and changing the display scale.

During IC layout you will frequently want to use the VIEW ALL command to view the complete drawing and then zoom in on a particular area to make a modification. The proper use of the VIEW LIMIT command can make this process very efficient.

Use the **ARRAY** command to limit display detail in arrays.

When the view limit is active, ICED™ limits the amount of detail it draws when displaying large areas. The ON and OFF keywords and the SCALE parameter are used to determine when the view limit is active. The remaining parameters determine how much detail will be displayed.

[ON | OFF] and [SCALE=*view_scale*]

The view limit is active when it has been turned on **and** the display scale (in pixels per user unit) is less than the critical *view_scale* set by the SCALE keyword. Notice that as you zoom out the view window becomes larger and the display scale becomes smaller. In a typical situation, you arrange things so that when you execute a VIEW ALL command, the view limit will become active. This will allow ICED™ to sketch your chip very quickly. However, when you zoom in on a small area to make modifications, the display scale will become larger and the view limit will become inactive. Then, ICED™ will draw all geometry in complete detail.

view_scale must be a real number in the range 0:32. The smaller *view_scale* is, the further out you must zoom to activate it. A *view_scale* of 0.0 means that the view limit will never be activated.

Example: **VIEW LIMIT ON SCALE=2**

For this example, let us say that your view window is 592 pixels wide (640 pixels - 48 pixels for the menu). Then the view limit will be activated when displaying at least:

$$\frac{592 \text{ pixels}}{2 \text{ pixels/user unit}} = 296 \text{ user units}$$

Example: **VIEW LIMIT ON SCALE=0.5**

We can look at this example in another way. If the current view window is 1220 user units (e.g. microns) wide, and the display is 592 pixels wide, the current display scale would be:

$$\frac{592 \text{ pixels}}{1220 \text{ user units}} = .485 \text{ pixels/user unit}$$

Since .485 is less than the 0.5 pixels/user unit scale set in the above example, the view limit would be active. However, zooming in only slightly would deactivate the view limit since the view scale would go over 0.5.

[DEPTH=*depth*]

The DEPTH parameter provides the most dramatic improvements in redraw time. The other VIEW LIMIT options require ICED™ to apply the view limit test to each component in the database to decide whether or not to draw it. The DEPTH parameter allows entire cells to be ignored when redrawing the screen.

The DEPTH parameter limits the depth to which ICED™ will display components in nested cells. When the view limit is active, ICED™ will draw only components whose depth is less than or equal to *depth*. *depth* must be a non-negative integer in the range 0:100.

All components in the root cell are said to have depth 0. Components contained in depth 0 cells are said to have depth 1. Components nested in depth 'n' cells have depth 'n+1'. The cells that form an array of depth n have depth 'n+1'.

[UNITS=*units*] and [DOTS=*dots*]

These parameters set limits on the size of the smallest object ICED™ will draw when the view limit is active. *units* must be a non-negative real number. *dots* must be an integer in the range 0:10. The UNITS parameter specifies a limit on the object's absolute size in user units. (Most users interpret 1 user unit as 1 micron). The DOTS parameter specifies a limit on the apparent size of the object in dots. (1 dot = 1 pixel on the screen.) For the purposes of these tests, the size of an object is the length of the longest side of its bounding box.

Example:

VIEW LIMIT ON SCALE=0.5 DOTS=10



Figure 115: View limit not active.

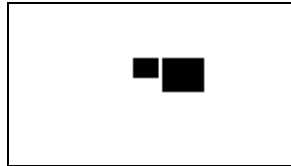


Figure 116: After zooming out, the view limit is active: one box smaller than 10 dots disappears.

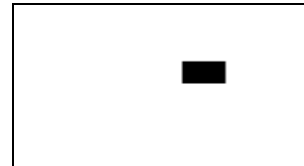


Figure 117: Zooming out further, two boxes are smaller than 10 dots.

The figures above show what happens as you zoom out, making the display scale smaller. In Figure 115, the display scale has not yet activated the view limit. The view limit becomes active in Figure 116, and since the smallest box is less than 10 pixels wide, it disappears. Zooming out still more, the middle box also disappears. Zooming in again, or using VIEW LIMIT OFF, will make the boxes reappear.

[SHOW_LAYERS=*layer_list*]

See **Layer Lists** in **Syntax Conventions**. The SHOW_LAYERS keyword is used to determine the layers that will be displayed when the view limit is active. Layers that are not on this list are not displayed. The *layer_list* does not affect the drawing of bounding boxes around cells and arrays. Placing layer 0 on (or removing it from) *layer_list* has no effect.

Example: **VIEW LIMIT ON SCALE=0.5 DEPTH=1 SHOW M1+M2+POLY**

After this command is executed, the view limit will be active whenever the view scale is less than 0.5 pixels per user unit. When this is the case, only objects on layers M1, M2, or POLY with depths of 0 or 1 will be drawn. (I.E. Only components in the current cell, or nested in cells added to this cell, will be displayed. Components nested more deeply than this will not be displayed.) The UNITS and DOTS parameters are unchanged by this command. If they have nonzero values from a previous VIEW LIMIT command, they will provide additional tests a component must pass to be drawn.

Example: **VIEW LIMIT SHOW_LAYER=1:***

This command places all layers on the SHOW_LAYER *layer_list*. None of the other VIEW LIMIT parameters are changed.

Example: **VIEW LIMIT**

This form of the VIEW LIMIT command (with no parameters) can be used to display the current values of the VIEW LIMIT parameters on the screen.

XSELECT

Enable or disable embedded selects in command files.

XSELECT (ON | OFF)

See the
@file_name
command for
details on using
command files.

The XSELECT command can be used to enable or disable embedded select commands in command files. (An embedded select command is generated when you execute a command that usually operates on selected components, but no components are currently selected.) The XSELECT mode is on by default enabling embedded select commands.

Consider the following command file fragment:

Example:

```
UNSELECT ALL  
SELECT LAYER "JUNK" ALL  
DELETE
```

If there are no components on layer JUNK, no components are selected when the DELETE command is executed. If the XSELECT mode is on, the DELETE command will execute an embedded SELECT NEAR command, bring up a near box, and wait for you to select something. The only way to bypass the embedded SELECT NEAR command would be to cancel the command, and this would cause ICED™ to issue an error message and leave the command file with any remaining commands unprocessed.

Example:

```
UNSELECT ALL  
SELECT LAYER "JUNK" ALL  
XSELECT OFF  
DELETE
```

In this example, the embedded select command is disabled since the XSELECT OFF command is included. Thus the DELETE command will do nothing when no components are selected. No error is generated and the remainder of the

command file is processed. The embedded select commands will remain disabled until you exit the command file or execute XSELECT ON.

The XSELECT mode has no effect on ICED™'s behavior when executing commands from the keyboard or menus. However, if you execute a XSELECT OFF command outside of a command file, embedded select commands will then be disabled by default within all command files. This can be overridden within a specific command file with a XSELECT ON command.

When you come to the end of a command file, ICED™ forgets the current XSELECT mode and restores the mode in effect when you entered the command file.

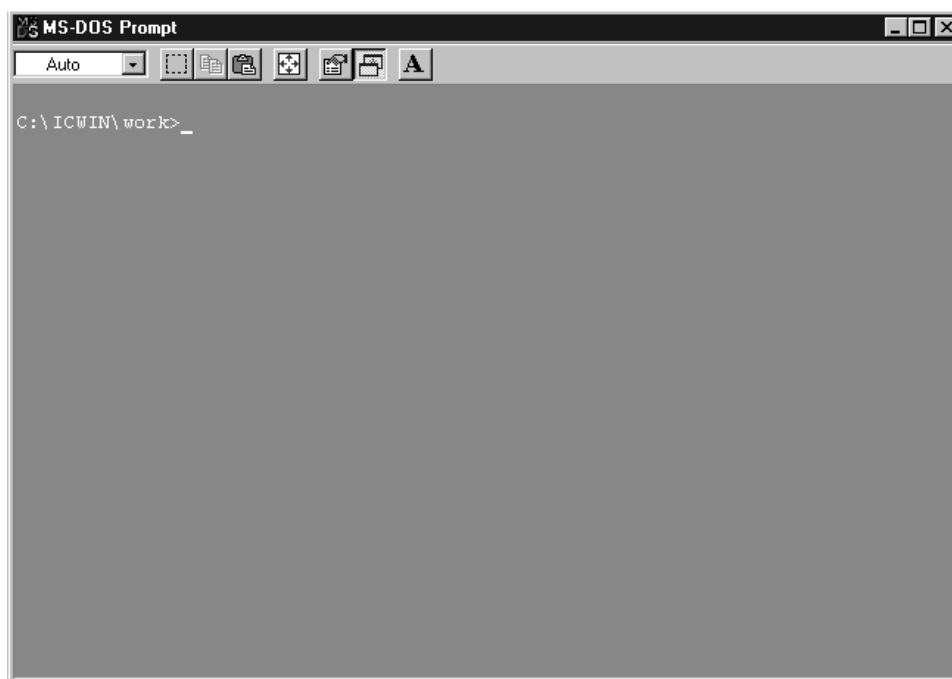
See page 99 to learn about journal files.

This following paragraph is for your information only. You may have noticed XSELECT statements in journal files. This form of the XSELECT command is intended for use in these files only. Other uses are **not** supported!

When you execute a command that generates an embedded select command, ICED™ must record the select information in the journal file. The following journal entry represents the embedded select command associated with a MOVE SIDE command.

```
XSELECT SIDE IN (-41.0, 27.5) (-39.0, 29.5)
MOVE SIDE BY (0.5, 6.0)
```

ICED™ Utility Reference



Overview

All of these utilities can be executed in a console window opened with the ICED desktop icon or with the SPAWN command in the layout editor. See page 371.

The ICED™ utilities perform a variety of tasks from printing plots to translating other layout file formats into ICED™ cells. The ICED™ utilities are not layout editor commands, and they are usually executed outside of the layout editor.

The **AllCells** utility prepares a batch file that will open ICED™ once for each cell in a library. When the batch file is executed, ICED™ will execute a specified command file in each cell.

Del2Cel is a utility to translate .DEL cell files created with older versions of ICED™ that did not support long file names.

ICList and **ICTree** provide information about the subcells nested in an ICED™ cell file. **ICList** lists all nested subcells in an alphabetical order, while **ICTree** presents the subcell names in a hierarchical listing.

ICTop is used to find top-level cells. It creates a list of cell names in the current directory that are not nested in other cells.

MkMenu is used to customize ICED™ menus.

MkPDF is used to make minor modifications to PDF files (Plotter Definition Files). The layout editor and the MkPlot utility use these files to build plots.

The **MkPlot** utility takes the .VEC file created by the ICED™ PLOT command and converts it to the format required by your printing or plotting device. MkPlot can send the output directly to your printer/plotter, create a print file, or create a bitmap file.

MkSti is used to create binary stipple pattern files for display and plotting. These files are used by the PATTERN and PLOT commands in ICED™. **UnMkSti** will read a binary pattern file and recreate its ASCII source file.

The **SFMap** utility is used to modify structure names in a Calma GDSII Stream file. This utility is especially useful for restoring mixed-case structure names in a Stream file.

The **UnStream** utility translates Calma GDSII Stream files into cell files. **SFDmp** will produce an ASCII dump of a GDSII Stream Format file. The **UnCIF** utility translates CIF (Caltech Intermediate Format) files into cell files.

There is at least one other .EXE file in your ICED™ installation. Files with names similar to **ICEDnAUX.EXE** are part of the ICED™ software licensing system. Do not rename or overwrite ICEDnAUX.EXE files or move them to a different directory.

You may or may not have .EXE files for other products from IC Editors, Inc., in your installation. These other products include the NLE, LVS, and DRC utilities. (See those manuals for details.) Other .EXE files may be auxiliary programs used by command files.

Executing ICED™ Utilities

These utilities are run as separate programs in a console window running the MS-DOS command interpreter just as the layout editor is executed from a console window. This console window can be a temporary one created only to execute the utility. Or the utilities can be executed at the command prompt in an open console window.

Once environment variables are defined in a console window, their definitions persist until the window is closed.

Some of the utilities require that various environment variables be defined in the environment of the console window before executing the utility. (See the table on the next page.)

There are three main methods for executing these utilities:

- 1) Type the utility command at the prompt in an open console window (often called an MS-DOS window).

- 2) Execute the utility command from the layout editor using the SPAWN command (or the DOS command if you want to wait for it to finish).
or
- 3) Include the utility command in a batch file.

Details on these three methods are covered in the following pages. The first and third methods may require that you define environment variables as described below. Using method 2 insures that the environment variables are already defined.

When Environment Variables are Required by Utilities

Project batch files and environment variables are more fully described beginning on page 39.

Several utilities require the same environment variables required by the ICED™ layout editor. The easiest way to define these environment variables is by executing the project batch file without arguments. Use the same project batch file you use to open the layout editor. This insures that the environment for the utilities is the same as the one used when you view or edit cells with the layout editor. This batch file is usually written to create the environment variables and then quit when executed without arguments.

If all relevant cell files are located in the current directory, ICList, ICTree and ICTop do not require ICED_PATH to be defined.

Environment variable	Utilities that use this variable
ICED_PATH	ICList, ICTree, ICTop, and the XCELL.BAT file created by the AllCells utility
ICED_USER	MkMenu, MkSTI, and MkPDF
DRC_PATH	The DRC the NLE programs and rules compilers
PATH	The operating system uses this list of directory names to find the executable files for all of the utilities without requiring the user to type the path to the file.

Figure 118: Utilities that require environment variables

Executing a Utility in a Console Window

It is best to execute some of these utilities when the layout editor is not running. Some utilities will steal significant computer resources from the layout editor. Utilities that create cells (e.g. UnStream and UnCIF) may create data that is in conflict with cells already loaded in the layout editor. Utilities that report on cell data (e.g. ICTop, ICTree, and ICList) use only cell data that is stored on disk and this data may be out of date for cells that have been modified in the current layout editor session.

Executing utilities like these from the Windows Start->Run prompt is not a good idea. No console environment is created in this case. Even the PATH environment variable (used by the operating system to locate the executable file for a utility) probably does not include the Q:\ICWIN²⁹ directory.

Opening a separate console window with the ICED desktop icon created by the installation has several advantages for executing utilities:

- The PATH environment variable is automatically modified to put the Q:\ICWIN directory in the front.
- The ICED_USER and ICED_HOME environment variables are automatically defined.
- If you want other customizations to the console environment to be made automatically, you can modify Q:\ICWIN\AUTOEXEC.BAT. This batch file is automatically executed as the last step in creating the new console window.
- Once you execute the project batch file (without an argument) in this console window, all other relevant environment variables are defined and will persist until the window is closed. (If you only use one technology and one project, you can execute the project batch file as a line in Q:\ICWIN\AUTOEXEC.BAT.)
- You can easily change the current directory with CD commands to any cell library or to a support directory.

Use the DOS command SET at a console prompt to see the current value of all environment variables.

²⁹ Q:\ICWIN represents the drive and path of the ICED™ installation directory. This is where all ICED™ utility executable files are stored

- You can execute as many utilities as desired in the open window, while working in other windows.
- You can repeat a utility by pressing the <↑> button since the DOSKEY system utility is automatically invoked.

Use the following method to execute utilities in a separate console window opened with the ICED desktop icon.

- 1) Click the ICED desktop icon.
- 2) In the open console window, execute your project batch file (i.e. the batch file you use to open the layout editor) with no arguments. This loads all of the environment variables defined in the batch file.
- 3) Use the CD (Change Directory) command to make your working directory the current directory. (If your project batch file includes this command, you do not need to execute it as a separate step.)
- 4) Type as many utility command lines as desired.
- 5) Close the console window by typing the EXIT command at the prompt or by using the close button (indicated with an 'X' in the upper right corner of the window.).

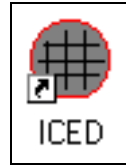


Figure 119: ICED desktop icon

If you need to repeat or correct a utility command line, you can usually repeat the last command typed at the console prompt with the <↑> key.

Executing a Utility while the Editor is Open

These commands can be triggered by function keys. See the **KEY** command.

The sample startup command file assigns the DOS command to the <F9> key.

Some utilities (e.g. MkPlot) are easily executed while the layout editor is open. For utilities like these, you can use the SPAWN or DOS commands from the editor command line. (These commands are fully documented in the command reference section of this manual.)

This method insures that any required environment variables are already defined. This is due to the fact that when you open the layout editor with the project batch file, a console window created by editor inherits the environment created by the project batch file.

The DOS and SPAWN commands can be executed only from inside the layout editor. The DOS command will open a console window and halt the layout editor until it closes after the completion of the utility. The SPAWN command is very similar except that the layout editor is not halted. You can continue to work in the layout editor while the utility executes in the console window.

If you include the utility command line in the SPAWN or DOS command, the utility is executed in a separate console window that will close automatically once the utility is complete.

Example:

SPAWN MKPLOT MYCELL /D /I

You can open a console window that stays open by typing the SPAWN command with no arguments.

Typing the proceeding command on the layout editor command line will execute the MkPlot utility to plot the MYCELL.VEC file. (MYCELL.VEC must already have been created by a PLOT command.) You can continue to work in the editor while the MkPlot utility executes, but if your plot is large and complex, you may see significant increases in response time in the layout editor. The MkPlot utility will cause the console window to stay open for a 20-second countdown after the operation is complete to allow you to note any error messages. Then the window will close automatically.

If you routinely execute other commands before executing a utility, you can combine other commands with the DOS or SPAWN command that executes a utility. Type these commands in a command file and execute all commands as a unit by executing the command file. You can even use interactive prompts or menus in this command file to control its execution. Refer to the overview on command files on page 14, the Command File Programmer's Reference Manual, and the sample command file Q:\ICWIN³⁰\AUXIL\PLOTNOW.CMD supplied with the installation.

³⁰ Remember that Q:\ICWIN represents the drive and path where you have installed the ICED™ program files.

On some operating systems, you can execute utilities in the console window used to open the layout editor while the layout editor is still open. Try typing in this window to test if it works on your system.

If you want to execute more than one utility command line, you can open a separate console window by using the SPAWN or DOS commands without any arguments. In this case, the console window will remain open until you close it with the EXIT command or close window button.

On Windows NT systems, the console window created by the DOS and SPAWN commands automatically inherits the environment already in place for the layout editor. Unfortunately, this is impossible on Windows 95 and 98 systems. For these systems, both the DOS and SPAWN commands use a batch file to set the environment. This file, Q:\ICWIN\TMP\W95\$ENV.BAT, is updated each time either command is executed. This process is automatic and transparent to the user.

Executing Utilities with a Batch File

Use the CALL command to execute one batch file from inside another.

A batch file containing a utility command line can be executed with any of the standard Windows methods (See a list on page 90.) You can include any valid MD-DOS commands.

When you need to execute a utility that requires environment variables, you can execute the project batch file to create the correct ICED™ environment directly from your utility batch file. When executed with no arguments, a project batch usually sets the environment variables and exits.

Example:

Close a console window with a title similar to "Finished..." by clicking the button with the 'X' in the upper right corner of the window.

```
CALL C:\icwin\myproj.bat  
E:  
CD \myproj\cell_lib  
C:\ICWIN\ICTOP /f > C:\icwin\tmp\ictop.txt
```

When a batch file containing the proceeding commands is executed, it will first execute the MYPROJ.BAT project batch file to create the appropriate environment variables required by the ICTop utility. The next two lines change the current directory to the desired cell library. Finally, the ICTop utility will execute. It will redirect the output to the file C:\ICWIN\TMP\ICTOP.TXT.

Interrupting ICED™ Utilities and Getting Syntax Help

Most of these utilities can be aborted by using the <Ctrl><Break> or <Ctrl><C> keyboard combinations.

Using "/h" as the only parameter will display a short help message on the syntax for most of the utilities.

Example: **ICTREE /h**

Type this command at a console prompt to get a brief listing of the syntax for the ICTree utility command line.

AllCells

Execute a command file on all cells in a library

AllCells

You can refer to the description of the _LOOP.CMD file in the Classroom Tutorial Manual for other methods of modifying multiple cells. However, these methods work only for cells actually nested in a design.

See the information beginning on page 371 to learn how to execute a utility. We do not recommend that execute this utility while the layout editor is already open.

AllCells.EXE is a relatively new utility to create a batch file that opens the layout editor to edit each cell in a library one at a time and executes a command file in each cell. The AllCells utility only creates the batch file (the name will be XCELLS.BAT); you must execute XCELLS.BAT to perform the processing. The batch file will open and then close the editor automatically for each cell, requiring no user interaction.

This type of processing is useful when you have to perform an operation on every cell in a library. For example, changing the layer numbers of existing geometry, reporting information on each cell, plotting each cell individually, etc.

As of this writing, this utility is not guaranteed to work on all operating systems. (Check Q:\ICWIN\DOC\CHANGES-LAY.TXT for recent improvements.) The batch file created by this utility may have problems running on the Windows 95, 98, and Me operating systems. This is due to the way these operating systems support running 32 bit applications in batch mode. The results of each editor process may not be reported correctly to the batch file process. (This results in an incorrect error file that records which cells failed the processing.) Also, multiple instantiations of the editor may overlap and interfere with each other. You can try this utility on these operating systems, but look over the results carefully to be sure each cell was processed correctly. You should not see any problems with the Windows NT, 2000, or XP operating systems.

If you have hundreds of cells in your library, this process may take some time. The layout editor is opening and closing for each cell. Since each new editor window pops up on top of your other windows, you will need to wait for the entire process to finish before using your computer for other purposes.

Any commands can be included in the command file. This includes all of the extra commands in the Command File Programmer's Reference Manual.

See page 45 for details on the cell library search path.

The answer to this prompt determines which command line option is added to each editor command line in the batch file.

See a comparison of termination commands on page 199.

The basic procedure to use AllCells is as follows:

- **Create the command file** to perform the desired processing. (We cover an example below.) Test the command file in a normal layout editor session on a sample cell to debug it.
- **Back up the cell library** first with any method so that you can recover from problems.
- Make the cell library the current directory in a console window with a **CD command**.
- If the cells in the library have cells nested inside of them that may be located in other cell libraries, you need to define the cell library search path for these nested cells. (No nested cells will be modified by XCELLS.BAT, only cells in the current library. However the editor will not be able to open any cell that contains a nested cell for which it cannot find a cell file.) The easiest way to define the ICED_PATH cell library search path is to **execute your project batch file** without the usual cell name argument. This will define the environment variables and then complete without opening the editor.
- Type **AllCells** in the console window.
- Respond to the prompt for the exit mode with one of the following keystrokes:
 - <E> EXIT mode The editor will close each time with an EXIT command that saves each cell file, overwriting each existing cell file.
 - <Q> QUIT mode The editor will close each time with a QUIT command that does not save the cell file. No existing cell files will be overwritten.
 - <L> LEAVE mode The editor will close each time with a LEAVE command that saves the cell file only if changes to the geometry have been made. Only cell files that were actually modified by the command file will be overwritten. (For example, if your command file deletes all shapes on layer X, cells that did have shapes on layer X will be overwritten with cells that have that layer removed, but cell files that did not contain any shapes on layer X will not have their contents replaced.)

<N> None mode No termination command will be added to the command file. Unless the command file contains a termination command, the editor will be left open every time and each editor session will need to be closed interactively. Whether or not the cell files are saved depends on the termination command you use to close each session.

- Next, respond to the prompt for the command file name with the **name of your command file**. If the command file is in the current directory, or in a directory on the command file search path defined in your project batch file, you will not need to type the path to the file.
- If this is the first time you are using this utility, look at the contents of the batch file the utility just created so you understand what the batch file will do. You can use the command **NOTEPAD XCELLS.BAT** to look at the file.
- Execute the batch file by typing **XCELLS** in the console window.
- If an error file was created, look at the names of cells that were not processed correctly. Type **NOTEPAD XCELLS.ERR** to open the file. Cells listed in this file were not saved.
- If the error file contains a list of cells, look at the end of each **cell.JOU** journal file to see the error messages. You can perform the processing individually on these cells, or correct the command file, delete the cell files in this directory, copy the cell files from your backup, and then execute **XCELLS.BAT** again.

Sample Command File To Change Layer Numbers

This example covers how to create a command file to change the layer numbers of existing geometry in every cell in a library. Suppose that you have a cell library where shapes on the M1 layer are stored on layer number 10. You now need to change layer M1 to be layer number 11 in all cells of your library.

Simply changing the definition of the M1 layer in each cell is not sufficient to make this change. You must change the layer number of all existing geometry on that layer.

See page 37 for details on why this type of command file is necessary when you change the layer name-layer number correspondence in a startup command file

This change requires that you also change the definitions of layers 10 and 11 in the startup command file. Layer number 11 now has the name M1 and layer 10 will have some other new name, or none at all. You need to execute the modified startup command file in each cell so that the new layer definitions are stored in each cell file. Let us assume that the name of this modified startup command file is NEW.CMD.

We will call the command file that should be executed in all cells to change the M1 layer M1NEWNUM.CMD. The contents of the file are shown below:

XSELECT OFF	! avoids problems when no components are selected when the SWAP command below is executed
UNBLANK ALL ; UNPROTECT ALL !	makes all components selectable
UNSELECT ALL	! avoids changing the layer of other components.
SELECT LAYER=10 ALL	! selects only all components on the M1 layer
SWAP LAYER 10 AND 11	! changes the layer number of all selected components
@NEW.CMD	! executes the new startup command file

Now that you have a command file to perform the layer number change, you would follow the steps on page 379 to execute these commands in every cell of your library. You should choose the Exit mode when executing the AllCells utility so that every cell file is saved after the command file completes. This way even cells that do not have shapes on layer M1 will have the new layer definitions stored in the cell files.

Del2Cel

Convert @...DEL files to .CEL files.

Del2Cel

See the information beginning on page 371 to learn how to execute a utility. We do not recommend that execute this utility while the layout editor is open.

This utility is intended for users of previous versions of ICED™ when cells with names longer than 8 characters had to be saved in cell files with unique names similar to *@ustring.DEL*. This utility converts each .DEL cell file to a cell file with the name *cell_name.CEL*.

If you are trying to launch ICED™ using a cell library that was used only by DOS-based versions of the layout editor, you may see the error message "Could not load cell *longcellname....*System reports no such cell or directory." This indicates that the cell file for at least one nested cell could not be located in the list of cell libraries defined in the project. If the error message is about a cell with a cell name longer than 8 characters, it may be that the cell is stored in a cell file with a .DEL extension. Executing this utility in all cell libraries with such cell files will allow you to open a cell that refers to these cells.

Once these cell files are converted, the new *long_cell_name.CEL* files are no longer compatible with DOS-based versions of the layout editor. **Be sure to copy your cell files to new libraries used only with Window-based versions of ICED™ before you run this utility.**

The Del2Cel utility will convert all .DEL files in the current directory to .CEL files. The .DEL files are not deleted.

Example:

DEL2CEL

When you execute this command, the utility will create a cell file with the name *long_cell_name.CEL* for every cell file in the current directory with a name similar to *@ustring.DEL*. Use the CD (Change Directory) command to change to the desired directory **before** executing the utility.

Since the cells that are nested inside other cells are added by cell name (not file name), the appropriate cell files will be found by the layout editor even though their cell file names have changed. You do not need to make any other changes to the cell files to make them compatible with the newer versions of ICED™.

ICList and ICTree

List cells nested in a cell file.

ICTREE [*[path\]cell_name*] [/d:*n1*] [/p:*n2* | /p] [/h]
ICLIST [*[path\]cell_name*] [/d:*n1*] [/p:*n2* /p] [/b] [/h]

Switches: /d:*n1* List cells to nesting **depth** *n1* (*n1* in range 1:100)
 /p:*n2* **Pause** screen display every *n2* lines (*n2* in range 10:99)
 /p Same as /p:22
 /b **Bare** listing of cell names only
 /h **Help**: list syntax message and exit

ICList and ICTree display the names of nested cells within a given cell file. You will be prompted for the cell name if you do not provide it. The operating instructions and command line options for both programs are almost identical.

See the information beginning on page 371 to learn how to execute a utility.

These utilities use the data stored in cell files. If cells have been modified in a current layout editor session and not yet saved, the changes will not be reflected in the reports generated by these commands. For this reason, it is best to execute these utilities in a console window when the layout editor is not open.

These utilities use the ICED_PATH environment variable to find cells in other directories. See page 45.

Both ICList and ICTree report the directory in which each subcell is stored. ICList creates an alphabetized list of subcells. In addition, ICList displays the ICED version that created each subcell and the number of divisions per user unit in each subcell. ICTree creates a hierarchical listing of the subcells using indentation to show their nesting within the drawing.

For examples on how these utilities work, assume you have a cell named PROTO.CEL. Subcells named ANDGATE and ORGATE are nested inside of PROTO. ANDGATE and ORGATE each have two subcells named PNP and NPN.

Example:

ICLIST PROTO

When this command is typed at the console prompt, while the directory where the cell PROTO is stored is the current directory, you will get an alphabetical listing on your screen similar to Figure 120. Note that the cell library paths are listed first and that the listing to the right of each cell name lists the path containing the cell by number. The div parameter refers to the number of divisions per unit. See page 76 for an explanation. The final parameter refers to the program (and version) that created the cell. "ICED-32" refers to the DOS-based versions of the ICED™ layout editor.

```
Subcell directories:
Path 0 = C:\BIPOLAR\CURRENT
Path 1 = C:\BIPOLAR\LIBRARY

PROTO      path=0 div=1000 ICED-32 v3.28
ANDGATE    path=1 div=1000 ICED-32 v3.28
NPN        path=1 div=1000 ICED-32 v3.28
ORGATE     path=1 div=1000 ICED-32 v3.28
PNP        path=1 div=1000 ICED-32 v3.28

Summary:
Cell PROTO contains 4 subcells
The maximum subcell nesting depth is 2
```

Figure 120: Sample output from ICList.

Example:

ICTREE PROTO

When the ICTree utility is used, you will get a hierarchical listing in your console window similar to Figure 121. The numbers in ()'s in an ICTree cell listing indicate the directory path number in which the cell was found.

Example:

ICLIST PROTO /d:1

This command stops looking for subcells after depth 1. It will list ORGATE and ANDGATE which are at depth 1. However NPN and PNP will not be listed since they are at depth 2. (All components in the root cell are said to have depth 0. Components contained in depth

```
ICTREE version 1.02 (C)Copyright 1992
IC Editors, Inc., ALL RIGHTS RESERVED

Subcell directories:
Path 0 = C:\BIPOLAR\CURRENT
Path 1 = C:\BIPOLAR\LIBRARY

PROTO(0)
ANDGATE(1)
  NPN(1)
  PNP(1)
ORGATE(1)
  NPN(1)
  PNP(1)

Summary:
Cell PROTO contains 4 subcells
The maximum subcell nesting depth is 2
```

Figure 121: Sample output from ICTree.

0 cells are said to have depth 1. Components nested in depth 'n' cells have depth 'n+1'. The cells that form an array of depth n have depth 'n+1'.)

Example: **ICTREE PROTO /p**

The /p flag causes ICTree to pause the screen display every 22 lines. This is useful when looking at long listings in a console window.

Example: **ICLIST PROTO /b > PROTO.LST**

This command uses output redirection to write the alphabetical list of cells to a file with the name PROTO.LST in the current directory. The listing will not appear on the screen, instead it will be stored in the file. You can then print or edit this file. If you are unfamiliar with redirected input and output, you may wish to read about it in your MS-DOS manual. Look in the index under "redirected ..." or "redirecting ...".

The /b flag is valid only with the ICLIST utility. When you are using output redirection to create a file containing cell names for input to another program, you can add the /b flag to get a Bare, or Brief, listing of only the cell names. No directory path listing or cell information is included. The root cell name is listed first, then all nested cell names in alphabetical order.

PROTO
ANDGATE
NPN
ORGATE
PNP

Figure 122:
Sample output
from ICList /b.

Use the DOS command SET at the console prompt to see the current value of all environment variables.

You should be aware that both ICList and ICTree use the environment variable ICED_PATH to define valid cell library directories. If you open the console window with the method shown on page 371, the ICED_PATH variable will be set correctly based on the settings in the project batch file. If you open your console window in another manner, and the cell libraries are not defined, you may get a message from either ICList or ICTree similar to the following:

******* There are 2 missing cells *******

This message will occur when you request a report for a cell that contains subcells that are stored in missing cell files. The search path for cell files is the current directory, then each of the cell libraries defined in the environment

variable ICED_PATH. If you get an error message similar to the one above, it is probably due to one of three reasons:

- Some cell libraries are omitted from the ICED_PATH definition, or this environment variable is not defined at all. Open the console window with one of the methods listed on page 371 to get the cell library list from the current version of your project stored in ICED_PATH.
- Some cell files are still in the *@string.DEL* format used by older versions of ICED™ for cells with long names. See the Del2Cel utility on page 378.
- Cell files are actually missing. This is not uncommon when opening cells imported from Stream or CIF files. Contact the person who gave you the import file to resolve these types of problems.

Since the search for nested cells requires that many cell files must be read, this utility may take a few moments to complete. Be patient and wait for the listing to display to your screen. (When redirecting the output, you can tell the command is complete when the console prompt reappears.) When some of your cell files are stored on networked drives, the wait may be longer.

ICTop

Find possible top-level cells

ICTOP [*path*] [/f] [/h]

Switches: /f **F**ull search of cell libraries for parent cells
 /h **H**elp: list syntax message and exit

This utility considers only cell files stored on disk.

Use this utility to find top-level cells. Top-level cells are cells that are **not** nested inside any other cell. This is useful when you are looking at cell files from another user and do not know what cell is the highest level cell that contains the design as a whole.

See the information beginning on page 371 to learn how to execute a utility.

Only cells in the root directory will be considered as candidates for top-level cells. The root directory is set by the *path* parameter if it is defined on the ICTop command line. When *path* is omitted, the root directory is the current directory.

Any child cell in the root directory that is nested inside any parent cell will be excluded from the listing.

The default is to check for parent cells only in the root directory. That means that if a child cell is nested inside a parent cell stored in a different directory, this will not be considered by the ICTop utility. All cells not nested inside parent cells stored in the root directory will be listed.

Example:

ICTOP

Typing the name of the utility with no parameters at the console prompt will produce a listing similar to Figure 123.

This example assumes that PROTO is the only cell in the current directory that is not nested inside another cell in the current directory.

Possible top level cells in C:\BIPOLAR\CURRENT:
PROTO

Figure 123: Sample output from ICTop.

Use the DOS command SET at the console prompt to see the current value of all environment variables.

When the /f flag is added to the command line, then the other directories on the environment variable ICED_PATH are searched for parent cells, but not for possible top-level cells.

The ICED_PATH environment variable will be set correctly based on your project settings if you open the console window with one of the methods shown on page 371. If you open your console window in another manner, and the cell libraries are not defined you will not get correct information using the /f flag.

This utility uses the ICED_PATH environment variable to find cells in other directories. See page 45.

Let us assume that you have received from another user a compressed file (e.g. a Zip file) that expanded into several cell libraries stored as subdirectories of your current directory. You need to determine what cell is the top-level cell that represents the design at the highest level (i.e. the whole chip). Your first step should be to create a new project batch file (from a copy of Q:ICWN.BAT) that defines all of the expanded directories as cell libraries in the definition of ICED_PATH. Then open a console window and execute this batch file to define the environment variable ICED_PATH. Use the DOS command CD to make the directory that contains all of the subdirectories your working directory.

Assume that the name of one of the libraries is CELL_LIB1.

Example:

ICTOP CELL_LIB1 /F

Type the command above at the console prompt and wait for the command to finish. Only cells in CELL_LIB1 are considered candidates for top-level cells. Cells in this library that are contained in any parent cell in any library will be eliminated from the list.

If the list generated contains only one cell, you have found the top-level cell. If the list contains no cells, then it is probably a supporting cell library that contains subcells only. Similarly, if the list contains many cells, then the library probably contains many subcells that are not used in the design. These cells may be left over from other projects.

If you have not yet located the top-level cell, type the command again to consider cells in a different library. Be sure to include the /f flag.

The search for parent cells when the /f flag is used requires that all cell files in all cell libraries must be read. This can take a while to complete. Be patient and wait for the listing to display on your screen. (When redirecting the output, you can tell the command is complete when the console prompt reappears.) When some of your cell files are stored on networked drives, the wait may be longer.

MkMenu

Create custom menus.

MKMENU [*path*] *file_name*

See the information beginning on page 371 to learn how to execute a utility.

This utility is used to create customized menus for ICED™.

The *file_name* parameter must be the name of an ASCII menu definition file with the file extension .DAT. If you specify a file extension other than .DAT, it will be ignored, and MkMenu will look for a file with an extension of .DAT anyway.

If you wish to make a few simple changes to the ICED™ menu structure, you can look at the file Q:\ICWIN\AUXIL\M1.DAT³¹. Copy this file to a new file name and make your changes in any ASCII text editor.

One important issue to keep in mind is that menus will appear different on different screen sizes and resolutions. If you create a long menu page, it may appear correctly on your screen, but fail to load on a computer with a lower resolution.

(This description will not provide the level of information you will need to create your own menus from scratch. If you need assistance in creating your own menus, please contact IC Editors, Inc. technical support.)

Once you have the .DAT file ready, run MkMenu. It will create the file *file_name*.MEN in the AUXIL directory.

It is important to execute the MkMenu utility from a console window opened with the ICED desktop icon, or from a console window opened with the SPAWN or DOS command in the layout editor. This will insure that the ICED_USER

³¹ Remember that Q:\ICWIN represents the drive and path where you have installed the ICED™ program files.

See page 44 to learn more about the search file path for auxiliary files.

environment variable is defined that will enable the utility to find the AUXIL directory path. If this environment variable is not defined, the menu file will be created in the current directory and you must copy it to the AUXIL directory yourself. The layout editor will not be able to load a menu file unless it is stored in the AUXIL subdirectory.

There are two ways to load the new menu. You can use the MENU command in ICED™ or in the startup command file. Or, use the MENU parameter on the ICED™ command line in your batch file

If you use the MENU command in your startup command file, you should be aware of how ICED™ loads menus. If a menu is specified with the MENU parameter in the ICED.EXE command line in the project batch file, that menu is loaded first. If this is a new cell, the startup command file is then executed. The menu specified by a MENU command in a startup command file will override any menu already loaded.

The menu file in use when you terminate ICED™ is saved in the environment database stored with the cell. The menu reference stored in an existing cell will be used by default in future edit sessions unless the MENU parameter in the ICED.EXE command line overrides it.

MkPDF

Modify a PDF file.

MkPDF *printer_name*

See the information beginning on page 371 to learn how to execute a utility.

This utility is used to create or modify PDF files (Plotter Definition Files) for use by the ICED™ PLOT command and the MkPlot utility. It can be used if you need to make minor changes to an existing plotter definition (e.g. a new page size) or for a new printer or plotter model that is very similar to an old one that already has a PDF file.

Existing PDF files can be found in the Q:\ICWIN-AUXIL directory.

To use this utility, follow these steps:

- Copy an existing PDF file, one similar to your printer or plotter, to the new file *printer_name*.PDF.
- Open a console window. (See the method for doing this on page 371.)
- Make the directory where the new *printer_name*.PDF file is stored the current directory in the console window by using the CD (Change Directory) command.
- Type MkPDF *printer_name* at the console prompt. Do not include the .PDF extension.
- Answer the questions the utility asks you until it is complete and the default console prompt reappears. The existing *printer_name*.PDF file is overwritten.
- Copy the *printer_name*.PDF file to the Q:\ICWIN³²\AUXIL directory.

See page 44 to learn more about the search file path for auxiliary files.

The MkPDF utility will read the PDF file and prompt you with each existing parameter in the file. If you do not want to change a given parameter, simply

³² Remember that Q:\ICWIN represents the drive and path where you have installed ICED™.

press <Enter> to use the existing value. Otherwise, type in the new value before pressing <Enter>.

Read about
STRIP
capability in the
description of
the **PLOT**
command.

The types of parameters you can change include printer resolution, supported paper sizes, default enable of STRIP capability, etc. Neither MkPDF nor MkPlot will verify that the values you enter are appropriate for your device. Incorrect values (particularly printer resolution) can lead to ruined plots. You should test your new PDF file by plotting with each size before releasing it to others.

Near the end of the utility program, MKPDF will prompt you to "Enter special flags: ". These flags have special meanings for each plotter type. A summary of the special flags in use at this time is given below.

Plotter type	Flag value	Meaning
EPSON ESCP2	8	Disable microweave. (Faster printing, more banding)
	16	Allow bi-directional printing. (Faster printing, vertical lines may be jagged)
HP LARGE FORMAT PLOTTERS (650C, 750C, etc.)	1	Disable PJI commands. MKPDF sends certain HP-PJI commands to ensure that Postscript capable plotters are in their HPGL/RTL mode. As far as we know, HP-PJI commands do not represent a problem to any RTL compatible plotters. However, they can be disabled by setting special flags=1.
HP COLOR DESK JET	2	Send plotter specific page size commands to plotter. (Required for HP1120C.)
HP PAINT JET XL	4	Send plotter specific page size commands to plotter. (Required for PaintJet XL.)
Hewlett Packard Acronyms:		PCL Page Control Language PJI Printer Job Language RTL Raster Transfer Language HPGL HP Graphics Language

Figure 124: Plotter flags used by MkPDF

To combine flags, add their values. Thus, if you want to set flags 8 and 16 enter the value 24.

See page 44 to learn more about the search file path for auxiliary files.

All plotter definition files should be located in the Q:\ICWIN³³\AUXIL directory to be used by the PLOT command or the MkPlot utility.

This description does not provide enough information to create a PDF file from scratch. If you need more assistance, contact IC Editors, Inc.

³³ Remember that Q:\ICWIN represents the drive and path where you have installed the ICED™ program files.

MkPlot

Print or plot .VEC plot files.

MKPLOT [*path*]*file_name* [*device_name* | /d] [/i] [/m:*m_bytes*] [/r:*dpi*] [/h]

Switches: /d Prompt user for *device_name* (useful in batch files)
 /i Allows utility to be launched from the ICED™ editor instead of the console
 /m Use up to *m_bytes* Megabytes of **m**emory (default is all available RAM)
 /r Set plotter **r**esolution to *dpi* dots per inch
 /h **H**elp: list syntax message and exit

MkPlot can now be run without exiting the editor. See the information beginning on page 371 to learn how to execute a utility.

This utility is part of the process to plot layout data from the ICED™ editor. The full process to create a plot on a printing device is as follows:

- Use the PLOT command while in an ICED™ layout session to select the area to be plotted and create the .VEC file.
- Execute the MkPlot command to translate the .VEC file into a form acceptable to your printer or plotter. (You can use a SPAWN command to execute the MkPlot utility from the layout editor command line.) You can have MkPlot send the translated data to your printer/plotter or have it create a print file on your disk.

If you specified a bitmap export in the PLOT command (e.g. the BMPCOLOR or BMPMONO plotting devices) then the .VEC plot data is translated into a .BMP bitmap file by the MkPlot utility.

You do not need to supply the .VEC extension with the *file_name* parameter when you use the MkPlot utility. If the .VEC file is in the current directory, you do not need to include *path*. If you do not include any parameters in the command, you will be prompted for *file_name*. However you will not be prompted for *device_name* (unless you add the /d switch).

When no *device_name* is defined, MkPlot will create a print file rather than send the data directly to a printing device.

If you include the *device_name* in the MkPlot command line, then the plot data is sent directly to that device. If you include the /d switch rather than including the *device_name* in the command, then you will be prompted for the device name. Both the *device_name* parameter and the /d flag will be ignored if the .VEC file specifies a bitmap export.

See
PLOTNOW-
.CMD for a
sample
command file
that executes
the MkPlot
utility with a
SPAWN
command.

If you want to execute MkPlot from the ICED™ editor with a SPAWN command (rather than at a console prompt) you should add the /i switch to the command line. This changes the way MkPlot executes in three ways:

- 1) If you have supplied *device_name*, the .VEC file is deleted after the plot has been successfully completed.
- 2) If *device_name* is busy, MkPlot waits until the device is free instead of terminating with an error.
- 3) MkPlot pauses 20 seconds after it completes its work allowing you to view any messages before the window closes.

The /m switch is used to limit the amount a memory available to the utility. Experimenting with this value may speed up your plot jobs. The default is 64 Megabytes..

When the device supports a range of print resolutions and you have not supplied a /r:*dpi* parameter, you will be prompted to enter the resolution (i.e. dots of ink/toner per inch of paper). If you are not sure what resolutions are supported, do not enter a value and let the prompt (if any) display your choices.

The resolution value is very important. Do not enter a value that your device does not support.

Plotting Directly to a Device

When you specify the *device_name* to MkPlot (either in the command, or by responding a prompt generated by the /d switch), the plot data will be sent directly to the printing device.

The device name may be a port (e.g. LPT2) or a network UNC device name (Universal Naming Convention: used to specify a file directory or device on another computer e.g. \\tuna\bigplots).

The primary advantage of using a device name is that MKPLOT will send the data directly to the plotter without making a large (possibly VERY large) .DOT file on the disk.

There are two possible disadvantages to sending the data directly to the printing device from the MkPlot utility:

- When writing directly to a device, MKPLOT computes a slice then sends the data for the slice to the plotter, then it computes another slice, and so on. Thus, the data is sent in blocks with a pause between blocks while MKPLOT computes the next slice. The network software may assume this pause is a sign of trouble and interrupt the plot with possible bizarre results (like a few feet of black output). Thus, if you want to use this option you should watch the first few plots carefully.
- The second problem is that the plotter may be busy for the sum of the computation and plot times instead of just the plot time, especially if your printer does not spool jobs. This may make you unpopular with anyone else who uses the plotter. (Fortunately, the computation time is generally short compared to the plot time.)

Example:

```
PLOT PLOTTER="LJET150" PATTERN="SAMPLE" SIZE=A ALL
SPAWN MKPLOT OPAMP LPT1 /M:10 /I
```

If you use the **SPAWN** command to execute a utility, you can continue to work in the layout editor while the utility is executing.

The two lines in the example above could be executed from the command line of the layout editor to create a plot and send it to a LaserJet printer attached to the LPT1 port. Assume the current cell name is OPAMP. The first command will create the file OPAMP.VEC. The second command will open a temporary console window to execute the MkPlot utility and translate the data in the file into appropriate print commands for LaserJet printers using 150dpi resolution. The plot data is translated into printer commands in slices and each slice of print commands is sent directly to the printer attached to the LPT1 port. The MkPlot command is limited to using 10 Megabytes of memory. This will probably slow the command considerably and make the data print in smaller slices. However,

more memory will be available to other applications while the plot is being created. After the translation is complete, the .VEC file is deleted automatically because the /I switch is included in the command line. The temporary console window is closed automatically after a 20 second delay.

Use of MkPlot when using a Network Printer/Plotter

Some older network software may misinterpret large plot jobs sent directly to a printer or plotter by the MkPlot command. MkPlot will send the job to the device in sections, pausing between sections. The network software is likely to time out during these pauses and assume that the print job has failed.

When using a networked device, you can use the method below to create the printer file with MkPlot, and then send the file to your printer or plotter as a separate step. This might be helpful if you are printing multiple copies.

Creating a Plot File

When you want MkPlot to create a print file rather than send the data directly to a printing device, do not specify *device_name* in the command. (These print files can get very large, if you prefer to avoid storing such a large file on your drive, use the direct option described above.)

Example: **MKPLOT OPAMP**

Typing this command at a console prompt will translate the data in the OPAMP.VEC file (already created by a PLOT command in the layout editor) into a print file that can be sent directly to a printer or plotter. The OPAMP.VEC file should already be in the current directory. The output file OPAMP.DOT will be created in the current directory. The *file_name*.DOT file is always created in the same directory as the *file_name*.VEC file. Insure that this drive has enough space to hold the print file.

MkPlot is aware which printer or plotter model you chose when you executed the PLOT command. Data consistent with the device chosen will be created in the *file_name*.DOT file. You can send the *file_name*.DOT file to your parallel port printer by typing a command similar to the following at the console prompt:

Example: **COPY /B OPAMP.DOT LPT1**

The /B switch on the MS-DOS COPY command forces DOS to copy the file as a binary file. This will prevent the COPY command from misinterpreting the carriage return characters.

Creating a Bitmap File

This version of ICED™ supports the export of bitmap graphics files. Follow the following steps to create a bitmap file:

You may create additional bitmap export devices with the MkPDF utility.

- In the layout editor, execute the PLOT command (the easiest way is to use the menu option FILE->PLOT) with the following options:

- ✓ For plotting device select a bitmap export device (e.g. BMPCOLOR or BMPMONO).
- ✓ Select the appropriate pattern file. (You can use a PATTERN or TEMPLATE command to see what pattern file is being used for the display.)
- ✓ For Super-Pixel mode, always use One-dot for monochrome (black and white) plots. You can use Four-dots for color plots, but this will decrease the plot resolution by half. (See the PLOT command for details.)

- ✓ For size, you choose between named sizes. The BMPMONO.PDF and BMPCOLOR.PDF plotter definition

Size label	Inches	Millimeters
AA	3.75 x 2.704	147.6 x 106.5
A	7.5 x 5.409	295.3 x 213.0

Figure 125: Default sizes available for bitmaps.

- files use the sizes shown in Figure 125. (Most document programs will allow you to change the size of the bitmap after you have placed it in a document.)
- ✓ No other parameters are required if you are using the menu option to generate the PLOT command. If you choose "None" for the remaining options, the entire cell is plotted at a scale to fit the size choice.

See page 371 to
for other
methods of
executing
MkPlot.

- Once the .VEC file is created by the PLOT command, use the SPAWN command to open a console window and execute a MkPlot command similar to the following example. Do not specify a *device_name* on the MkPlot command line. You must choose the resolution of the bitmap at this point.

Example:

SPAWN MKPLOT OPAMP /R:60 /I

Type the above command right on the layout editor command line. This command will create a 60dpi bitmap graphic file with the name OPAMP.BMP. This file is suitable for including in a document created with a program like Microsoft WORD or Adobe Framemaker. The resolution of the bitmap will be 60 dots per inch. If the /r switch was not included, you would have been prompted for the resolution value.

If you are creating a bitmap file, it is best to enter a resolution value that matches the resolution of the printer that will eventually be used to print the graphic. Resolutions larger than this value will make the file larger but will not increase the sharpness of the printed image (unless you intend to scale the image). Extremely high resolutions will require large amounts of system resources (i.e. memory and file size.)

MkSti and UnMkSti

Prepare stipple file for fill pattern creation.

MKSTI [*path*] *file_name*

UNMKSTI [*path*] *file_name*

The .STI file is also used by the **PLOT** command.

ICED™ can display components using solid fill, no fill (outlines only), or user definable stipple patterns. The definitions for the stipple patterns are stored in a stipple pattern file (.STI file). The MkSti utility (executed from the console prompt or a batch file) is used to create the .STI file. Once inside ICED™, the PATTERN command is used to load the stipple file into memory, and the LAYER command is used to assign specific patterns to individual layers. The FILL command is used to enable or disable pattern fill for display.

See the information beginning on page 371 to learn how to execute a utility.

The MkSti utility reads an ASCII input file with a .DAT extension and creates a binary output file with the same name but with a .STI extension. UnMkSti will read a binary .STI file and recreate the ASCII source file that created it. You can then edit this file with an ASCII text editor and run MkSti on the modified file to create a modified stipple pattern file.

The .DAT file consists of a sequence of comment, pattern number, and stipple pattern lines. The first non-blank character in a comment line must be an exclamation mark (!). Blank lines and comment lines are ignored by MkSti. You can insert them anywhere in the .DAT file.

Each pattern starts with a pattern number line. The first non-blank character in the pattern number line must be a pound character (#). The # is followed by a pattern number in the range 2:100. (Pattern 0 is always reserved for a blank pattern (no fill) and pattern 1 is reserved for a solid fill pattern.)

The pattern number line is followed by pattern lines that represent an array of pixels. Each stipple pattern array should have 1, 2, 3, 4, 6, 8, 12, or 24 columns and must consist of 'X's and '.'s used to mark filled and unfilled pixels respectively. Blanks are ignored. You can provide 1, 2, 3, 4, 6, 8, 12, or 24 rows for each pattern array. The array supplied is tiled (i.e. repeated) to fill outlines.

The ICED™ distribution includes the file Q:\ICWIN³⁴\SAMPLES\SAMPLE.DAT. SAMPLE.DAT is a sample input file for the MkSti utility. The first few lines of that file are shown in Figure 126. These examples are defined with 12 by 12 arrays of pixels.

Note that fill patterns 2 and 3 in Figure 126 are very similar, but slightly offset from each other. They were created this way so that when a component drawn with pattern 2 overlaps a component drawn with pattern 3, the fill patterns (and colors) for both components will be visible. The fill patterns will not be exactly on top of each other. This is a good concept to keep in mind when creating your own fill patterns.

Example:

```
COPY Q:\ICWIN\SAMPLES\SAMPLE.DAT
REN SAMPLE.DAT MYPAT.DAT
NOTEPAD MYPAT.DAT
MKSTI MYPAT
```

```
#2
X . . . . X X . . . . X
. . . . X X . . . . X X
. . . . X X . . . . X X .
. . X X . . . . X X . .
. X X . . . . X X . . .
X X . . . . X X . . . .
X . . . . X X . . . . X
. . . . X X . . . . X X
. . . X X . . . . X X .
. . X X . . . . X X . .
. X X . . . . X X . . .
X X . . . . X X . . . .

#3
. . X X . . . . X X . .
. X X . . . . X X . . .
X X . . . . X X . . . .
X . . . . X X . . . . X
. . . . X X . . . . X X
. . . X X . . . . X X .
. . X X . . . . X X . .
. X X . . . . X X . . .
X X . . . . X X . . . .
X . . . . X X . . . . X
. . . . X X . . . . X X
. . . X X . . . . X X .
```

Figure 126: Portion of SAMPLE.DAT, a sample input file for MkSti.

³⁴ Remember that Q:\ICWIN represents the drive and path where you have installed the ICED™ program files.

See the information beginning on page 371 to learn how to execute a utility.

When the first three lines on the previous page are executed from a console window opened with the ICED icon they will create copy of the sample pattern source file in the current directory with the name MYPAT.DAT, and then the Notepad text editor is opened to modify the file. Once the modified file is saved, the next command will execute the MkSti utility. The Q:\ICWIN\AUXIL\MYPAT.STI pattern file will be created.

See page 44 to learn more about the search file path for auxiliary files.

It is important to execute the MkSti utility from a console window opened with the ICED desktop icon, or from a console window opened with the SPAWN or DOS command in the layout editor. This will insure that the ICED_USER environment variable is defined that will enable the utility to find the AUXIL directory path. If this environment variable is not defined, the pattern file will be created in the current directory and you must copy it to the AUXIL directory yourself. The layout editor will not be able to load a .STI file unless it is stored in the AUXIL subdirectory.

UnMkSti always creates its output file in the current directory. You should copy the created file to the AUXIL subdirectory.

SFDmp

Create an ASCII dump from a GDSII Stream Format file.

SFDMP [*path*\] *sf_file_name* [/p=*position*]

The SFDmp utility will create a dump of a Calma GDSII-Stream Format file in a format that can be read using any ASCII text editor. This can be useful if the UnStream utility or a different program reports a problem with a Stream file.

See the information beginning on page 371 to learn how to execute a utility.

The *path* parameter is optional. If used, you can also define a drive letter in *path*. If *path* is not used, SFDmp will look in the current directory for the Stream file.

sf_file_name specifies the name of the Stream file to be read. The file extension of .SF is not required. If your file uses a different extension specify it in the *sf_file_name* parameter.

The dump will be created in the same directory as the Stream file. The file name will be *sf_file_name*.DMP.

When dumping an entire Stream file, the size of the dump is usually at least three times as large as the Stream file. Be sure that you have enough available disk space before using SFDmp. You will also need a utility that can handle browsing large ASCII files to view it.

If you prefer to list only a portion of the file around a known file position, use the /p=*position* option. This option limits the output to a region of roughly 4 Kbytes positioned around *position*. Most programs that read Stream files will report errors with a file position where the error was found.

Example: **SFDMP TOPCELL /P:12949**

This command will read the file TOPCELL.SF in the current directory and dumps records starting between file positions 10901 and 14997. The dump is written to the file TOPCELL.DMP in the current directory.

It is not unusual to have a null record at the end of a stream file, so do not be alarmed if the SFDmp utility reports this when it terminates. The null record is not listed in the dump.

SFMap

Change GDSII Stream file structure names.

SFMAP *[[path\]sf_file_in* *[[path\] sf_file_out]*] (*[[path\]map_file* | /l | /u | /h)

- Switches: /l Translate all structure names to **L**ower case.
 /u Translate all structure names to **U**pper case.
 /h Display help text and exit

You can execute this utility at a console prompt. See page 371.

Or you can execute it with a SPAWN command in the editor.

This utility will read a GDSII Stream file and create a new Stream file with modified structure names.

You must use a different file name for *sf_file_out* than the input file name.. When you do not supply the file names in the command line, you will be prompted for them. If the files are in the current directory, you do not need to supply the directory paths to the files. You do not need to supply the file extension in the file name parameters. The program will assume a file extension of .SF for the Stream file name parameters and .MAP for the map file.

You will not need a map file if you use one of the switches.

Example: **SFMAP STREAMIN STREAMUP /U**

Typing the command above in a console window will read the file STREAMIN.SF in the current directory. All structure names will be converted to upper case and the file will be saved as STREAMUP.SF in the same directory.

Example: **SFMAP /L**

The command above will prompt you for the input and output Stream file names. The input file will be read, all structure names will be translated to lower case, and the modified Stream file will be created with the output file name.

If you need to make more detailed changes to structure names, you will need to prepare a map file before you execute the utility.

If you need to translate only a few structure names, you can create the map file with any ASCII text editor. Each structure name mapping should be on a separate line. You can add blank lines, and add comments preceded by a n exclamation mark (!). Each mapping line should be of the form:

M: *input_structure_name output_structure_name*
or
M: *input_structure_name*

The second form is intended to avoid warning messages when the *input_structure_name* needs no translation. (In this case the *input_structure_name* will be used in the output file.) The SFMap utility will generate a warning when a map file is used but there are structure names in the Stream file that are not present in the map file,. The stream file is still created, but if you prefer to avoid the warnings, add lines of the second form above so that all structure names are listed in the map file.

For example, suppose that your Stream file contains the structure name “INVERTER1X” and you want to translate this structure name to “Inverter1X”. Suppose also that your design contains only this structure and one other with the name “MYCELL”. MYCELL needs no translation.

The map file for our example would be an ASCII file with the following lines:

```
! Structure mapping file for MYCELL  
M: MYCELL  
M: INVERTER1X Inverter1X
```

Let us say the name of this file is MYCELL.MAP and the original Stream file name is MYCELL.SF. Typing the following command at the command prompt in the layout editor will translate the file and place the output in MYCELL_OUT.SF.

Example: **SPAWN SFMAP**

Special Characters in Structure Names

The SFMap utility allows structure names to contain blanks or a variety of special characters, even unprintable ones. Precede each special character with a backslash (\).

!Sample map line		Replaces this	with this
M: FLIP_FLOP	flip\ flop\!	! "FLIP_FLOP"	"flip flop!"

Since the exclamation mark (!) is treated as the comment indicator in map files, you need to precede it with a backslash to use it in a structure name as shown in the sample map line above. To use a comma (,) or a backslash (\) in a structure name, you must also precede them with a backslash.

!Sample map line		Replaces this	with this
M: NAND_2	Nand\,2X	! "NAND_2"	"Nand,2X"
M: NAND_2_3	Nand\\2X\\3X	! "NAND_2_3"	"Nand\2X\3X"

To use non-standard or unprintable characters in a structure name, use a backslash followed by three octal digits.

Using SFMap in Combination with UnStream

The SFMAP utility is particularly useful in combination with the UnStream utility. Let us say that you have received a Stream file that contains many non-standard characters in structure names from a third party. You need to return a Stream file to this third party and they want the non-standard structure names used in your Stream file. The general procedure to accomplish this is as follows:

The alias file must be edited to map at least one cell name for UnStream to automatically create a map file.

- 1) The first pass of UnStream warns you to edit the alias file to convert non-standard structure names. (If your structure names are in lower case, or mixed case, be sure to use the default reply at the "Is Case Significant?" prompt.)
- 2) Edit the alias file to supply a valid cell name for each non-standard structure name.

- 3) The second pass of UnStream uses the alias file and automatically creates an appropriate map file in addition to the cell files.
- 4) Modify the design as needed in the ICED™ layout editor.
- 5) Export the design into a Stream file with the STREAM command in the layout editor. The cell names are used as the structure names.
- 6) Execute SFMap with the map file created by UnStream to translate the structure names back to what they were originally.
- 7) If there were new structures added while you modified the design, then there will be warning messages about missing structure names produced by SFMap and recorded in the file *map_file.ERR*.
- 8) Add lines for the missing structure names to the map file. And re-execute SFMap if desired.

For example, assume that the original Stream file was named DESIGN.SF. Then when you imported this stream file, the first pass of UnStream reported non-standard structure names that needed to be mapped to valid cell names in the alias file. You did this and the second pass of UnStream created the file DESIGN.MAP.

When you began editing your version of the layout, you changed the name of the highest-level cell to MYDESIGN. When you are ready to export MYDESIGN to a Stream file you use a command file to automate the process. The command file contains the following lines.

Example:

```
@STREAMNUM  
STREAM MAP  
SPAWN SFMAP MYDESIGN MYDESIGN_FINAL DESIGN
```

The first line calls another command file that defines Stream layer numbers for all ICED™ layers. (This is a required step in Stream export. See the STREAM command.) The second command creates the Stream file MYDESIGN.SF. The third line executes the SFMap utility to translate the structure names back to the original non-standard format.

However, since you added a new structure name that was not in the original design, SFMap creates a file DESIGN.ERR with a warning that the structure MYDESIGN was not listed in the map file. Just add the following line to the map file DESIGN.MAP, and then execute the entire command file again to successfully create the Stream file with no warnings.

M: MYDESIGN

UnCIF

Convert CIF files into ICED™ cell files.

UNCIF [*path*] *file_name*

It is best to execute this utility at a console prompt when the layout editor is not running. See page 371.

The UnCIF utility translates CIF (Caltech Intermediate Format)³⁵ files into equivalent ICED™ cell files. This translation is done from a console prompt, outside of an ICED™ session.

The *file_name* parameter is the name of the CIF file to translate. This file must have a .CIF file extension. Any different extension you supply as part of *file_name* will be ignored. You can specify an optional *path* parameter if the CIF file is not in your current directory. All output files created by UnCIF are stored in the current directory. (We recommend using a new temporary directory, so no existing cell files are overwritten.)

You can use the **LIMIT=ON** command line parameter when you launch ICED™ to view a large dense top-level cell created by UnCIF. This will result in a much faster and easier to see initial display.

The translation is done in three steps.

1. The first pass of the UnCIF utility parses the CIF file syntax and prepares an alias file.
2. The alias file should be edited to adjust the symbol/cell name correspondences and layer assignments.
3. Execute the UnCIF utility again. This second pass of UnCIF performs the translation and cell file creation.

The alias file contains parameter lines that direct the translation of the CIF database during the second pass. These lines tell UnCIF how to map layers in the CIF file into layers in the ICED™ cell files and how to name the ICED™ cell files.

³⁵ The CIF standard is defined in "An Introduction to VLSI Systems" by Lynn Conway and Carver Mead, Addison-Wesley Publishing Company, 1980.

The alias and cell files will be created in the current directory even if the CIF file exists in a different directory. You should edit the alias file with a text editor before executing the second pass. (In fact, unless you provide the first pass with a layer correspondence file, you **must** edit the alias file. This is covered in more detail below.)

When the UnCIF command is executed, it looks for the alias file in the current directory with the name *file_name*.ALI. If this file does not exist, UnCIF executes the first pass to create it. If the alias file does exist, UnCIF will execute the second pass which creates the cell files. An ICED™ cell file is created for each symbol in the CIF database.

The first pass will prompt you for two pieces of information. UnCIF will provide defaults for both prompts in brackets []'s. To use the default provided, just press <Enter> at the prompt.

Enter prefix for default ICED cell names [SYMBOL]

This string will be used as the default for the top level cell name as well as the prefix for all unnamed cells. The string should not contain characters that would result in invalid file names. You will be warned if you use illegal characters.

Enter optional layer file name [NONE]

The layer file must consist of valid CIF - ICED layer correspondences. The syntax of these layer lines is covered below. If you do not have a layer file prepared, simply press <Enter> to use the default of NONE. In this case, you **must** edit the layer correspondences in the alias file.

Alias File Format

The alias file for UnCIF is composed of comments, a top line, symbol lines, and layer lines. Comments consist of an exclamation mark (!) followed by text. Comments may be placed on separate lines or at the end of parameter lines. The UnCIF utility ignores comments.

The top line defines the name of the top level ICED™ cell UnCIF will create. CIF files are usually constructed with an unnamed top level symbol consisting only of one reference to the highest level symbol containing data, so you can usually delete this cell after viewing it. The top line should consist of the character 'T:' followed by the cell name. The default prefix you defined for cell names will be used by UnCIF to create the top level cell name in the alias file. If for some reason the name provided is unusable as a ICED™ cell name, UnCIF will create a top line in the alias file with asterisks (****) instead of a cell name. You **must** replace the asterisks with a valid cell name using a text editor before using the alias file in the second pass of UnCIF.

Symbol lines control the mapping of the CIF symbol numbers into ICED™ cell names. Symbol lines consist of the characters 'S:' followed by a CIF symbol number and the corresponding ICED™ cell name. The first pass of UnCIF creates a symbol line in the alias file for each symbol in the CIF file.

If the CIF file uses a standard user 9 command to name a symbol, UnCIF uses the string in that command as the cell name. If the string in the user 9 command contains blanks, the symbol line created by the first pass of UnCIF will not be read properly by the second pass. You should look at the alias file carefully before the second pass and correct any undesirable cell names.

When the symbol name is not provided in the CIF file, the first pass of UnCIF will create a unique cell name consisting of the prefix you provided, followed by a pound symbol (#) and the CIF symbol number. You **may** replace the default cell names provided by the first pass with cell names of your own choosing. You **must not** change any of the CIF symbol numbers or delete any of the symbol lines. This will result in translation errors in the second pass.

Layer lines control the mapping of CIF layer names to ICED™ layer numbers. The layer lines in the alias file (or in the optional layer correspondence file) consist of the characters 'L:' followed by a CIF layer name and then a valid ICED™ layer number. Unless a layer file is used by the first pass of UnCIF, it will create layer lines in the alias file with question marks (?) instead of ICED™ layer numbers. You **must** replace all question marks in the layer lines with valid ICED™ layer numbers before using the alias file with the second pass of UnCIF. Only layer numbers, not layer names, can be used.

If you wish to prepare a layer correspondence file for input to the first pass, use any ASCII editor to create default layer correspondences using the syntax explained above. Create the file with a .LAY extension. (You can copy lines from an old alias file to save time.) Any lines from the .LAY file with CIF layer names used in the CIF file will be copied into the alias file. Layers which do not appear in the CIF file will simply be ignored. Whether or not the .LAY file is used, the first pass of UnCIF will create layer lines in the alias file only for layers actually used in the CIF file.

If you do not prepare a .LAY file, you must edit the alias file every time you translate a CIF file. Preparing and saving a .LAY file can save you time if you will be translating many CIF files of the same technology.

You can later associate layer numbers with layer names in the ICED™ cell using the **LAYER** command. This is usually done in the startup command file.

The following is a sample alias file created by the first pass of UnCIF. No .LAY file was used. Note the '?'s on the layer lines. They **must** be replaced with valid ICED™ layer **numbers** before executing the second pass. The top-level cell uses the default cell name 'SYMBOL'. In addition to the top-level cell, one cell named 'NN' would be created from the CIF symbol labeled with the number '1'.

```
S:1      NN
T:       SYMBOL
L:CAA    ?
L:CMF    ?
L:CCA    ?
```

CIF File Translation Once the Alias File is Created

The second pass of UnCIF performs the translation. It creates a cell file for every symbol in the CIF file. These cell files are stored in your current directory, even if the CIF file is stored elsewhere. If there are existing cell files with the same file names, the existing files will be destroyed. For this reason, **we strongly suggest that you use UnCIF in a separate working directory.** After successful conversion of the CIF file, you can copy the cell files into the appropriate directories.

To start the second pass, repeat the UnCIF command, with the same *path* and *file_name* parameters, in same directory where you executed the first pass of UnCIF.

When UnCIF executes, it first searches for the alias file, *file_name.ALI*. If it finds this file it will automatically start the second pass using the alias file.

The second pass of UnCIF provides you with several options about how to handle your file. Most CIF databases will be read properly using the defaults for each option. These defaults are indicated by square brackets, [], in the prompt. To use each default, simply press the <Enter> key.

The options UnCIF will prompt you for are described below.

Enter scale factor in centi-microns per user unit [100]:

If each user unit in the ICED™ cell files is intended to represent 1 micron, then use the default for this prompt. If you prefer to have 1 user unit represent 1 mil, type 2540 in response to this prompt.

The CIF standard uses units of centi-microns. The above prompt assumes that you are using this standard. If your CIF file uses a different unit of measurement, substitute that unit for the term "centi-micron". For example, if your CIF file uses a lambda ($\frac{1}{4}$ micron) as the basic unit of measurement, you can assume that the prompt reads "Enter scale factor in lambdas per user unit:". In this case, if one ICED™ user unit represents one micron, you would enter '4' in response to the prompt.

Enter divisions per unit [1000]:

If no NDIV parameter is set in the ICED™ command line, the default is 1000. See page 76.

UnCIF will ask you for the number of divisions per ICED™ user unit. Use the default unless you are sure that you know what you are doing. (If you do not use the default of 1000, you must specify the same number in the NDIV parameter on the ICED.EXE command line when you launch the editor to open these cell files.)

$$1 \text{ database unit} = 1 \text{ user unit} / \text{NDIV}$$

The database unit is the smallest unit of measure that ICED™ can use when storing coordinates. For example, if user units are microns and divisions per user unit is set to 1000, then all coordinates in a cell file must be aligned to a 0.001 micron grid. We strongly recommend using at least 100 divisions per unit.

Enter default text height [2.0]:

The response to this prompt will be used as the height of text components translated from the CIF file.

Note on Editor Settings of the Imported Cells

All cells created by this utility have no environment database. For example, when a new cell created with UnCIF is opened for the first time there will be no layer definitions, no grid definitions, and only a default resolution for coordinates.

For these reasons, when a cell created by UnCIF is opened for the first time, ICED™ will execute the startup command file to define these types of settings. (Usually, the startup command file is executed only when you are creating a cell that does not yet exist.)

Be sure you have a startup command file defined for your project when you open your newly imported files or your cells will not have settings such as layer name definitions. Refer to page 25 to learn more about startup command files.

UnStream

Convert GDSII Stream Format files into ICED™ cell files.

UNSTREAM [*path*] *sf_file_name* [/d]

It is best to execute this utility at a console prompt when the layout editor is not running. See page 371.

The UnStream utility translates Calma GDSII-Stream files into equivalent ICED™ cell files. This translation is done from a console prompt, outside of the ICED™ program.

The *sf_file_name* parameter is the name of the Stream file to translate. The file extension will default to .SF if you do not supply it with *sf_file_name*. You can specify an optional *path* parameter if the Stream file is not in the current directory. However, all output files created by UnStream are stored in the current directory. We recommend using a new temporary directory so no existing cell files are overwritten.

The translation is done in three steps.

1. The first pass of the UnStream utility parses the Stream file syntax and prepares an alias file.
2. The alias file can be edited to adjust the structure/cell name correspondences and layer assignments.
3. Execute the UnStream utility again. The second pass of UnStream performs the translation and cell file creation.

The alias and cell files will be created in the current directory even if the Stream file exists in a different directory. You can modify the alias file with any ASCII text editor before executing the second pass.

The syntax for both UnStream passes is exactly the same. When the UnStream command is executed, it looks in the current directory for an alias file with the name *sf_file_name*.ALI. If this file does not exist, UnStream executes the first pass to create it. If the alias file does exist, UnStream will execute the second

pass which creates the cell files. An ICED™ cell file is created for each structure in the Stream database.

Both passes require that the user respond to questions about how the translation should proceed. Add the /d switch to the command line to use the defaults for all of these options. When /d is added to the command line, each pass will execute without user interaction. This is useful in batch files.

The First Pass of UnStream

When UnStream is executed and no *sf_file_name*.ALI file exists, it will perform a first pass of the data and create the *sf_file_name*.ALI file.

Unless you have added the /d option to the command line, this first pass will prompt you for several parameters. To use the default in each case, simply hit <Enter> at each prompt.

Enter optional layer file name [NONE]:

This prompt allows you to supply the name of a file with layer correspondences. This file is used to map stream layer numbers to ICED™ layer numbers. By default, UnStream assumes that mask layers are numbered identically in the Stream and ICED™ databases (**except** that Stream layer 0 will be mapped into ICED™ layer 255). If this is not adequate for your Stream file, there are two ways to change how the second pass will handle layer correspondence. The first method is to provide the optional layer correspondence file for the first pass. (We will cover how to create this file on page 422.) The second method is to edit the alias file after the first pass is complete.

To allow the translation to continue without a layer file, just press <Enter>.

The SFMap utility can be used to modify structure names (including mixed case) in a Stream file. See page 407.

Is case significant in structure names ([Y] or N):

UnStream now supports mixed case structure names. When you use the default by pressing <Enter>, you will be able to use the SFMap utility at a later time to restore the case of structure names when exporting a new Stream file. Unless you have a reason not to, always use the default. Your response is saved in the stream file in a Case line beginning with the string "C:".

Enter optional cell suffix [NONE]:

When you do provide a cell suffix, it will be added to the end of each structure name when generating the ICED™ cell names.

Alias File Format

You can type the NOTEPAD.EXE command line right at the console prompt.

Once the first pass is complete, edit the new alias file with a text editor. The alias file is composed of comments, a case line, structure lines, and layer lines. Comments consist of an exclamation mark (!) followed by text. Comments may be placed on separate lines or at the end of structure or layer lines. The UnStream utility ignores comments.

The Case Line

The alias file must be edited to map at least one cell name for UnStream to automatically create a map file.

The Case line has the format "C: *case_flag*". If *case_flag* is set to 1, case is ignored in structure names and all names are translated to upper case. When *case_flag* is set to 2, case is preserved in the map file. This map file can be used later with the SFMap utility to restore the original case of structure names.

Structure Lines

Structure lines control the mapping of the Stream structure names into ICED™ cell names. Structure lines consist of the characters 'S:' followed by a Stream file structure name, one or more blanks, and the corresponding ICED™ cell name. The first pass of UnStream creates a structure line in the alias file for each structure in the Stream database.

Wherever possible, UnStream uses the structure name as the cell name. If the character '?' appears in a structure name, it is replaced by a '#' in the cell name. This is necessary because a question mark acts as a wild card in filenames.

Since blanks are invalid in cell names, each blank space is replaced with a '_' in the cell name. Since blanks in the original structure names will cause problems for the alias file parser in the second pass, each blank in a structure name is replaced with a '^' symbol in the alias file.

To convert structure names back to their original names when exporting cells imported with modified names, see the **SFMap** utility.

When the structure name cannot be converted to a legal ICED™ cell name, a row of asterisks ("****") will be used instead of the cell name. You **must** use a text editor to replace any such row of *'s with a valid ICED™ cell name before executing the second pass of UnStream. You **may** also replace the default cell names provided by the first pass with cell names of your own choosing. You **must not** change any of the Stream structure names or delete any of the structure lines. This will result in translation errors in the second pass.

(If you think you might want to use the SFMap utility on an exported Stream file at a later date to restore the original structure names, changing the cell name in any structure line will result in the creation of a map file that you can use as an input file for SFMap.)

Structure names may have as many as 100 characters. However, to use names this long, you must edit the alias file, since ICED™ cell names can have no more than 32 characters.

The following is a fragment of an alias file with a set of structure lines before and after editing. This is a portion of the alias file created by the first pass:

```
S: CELLNAMELONGERTHAN32CHARACTERSABC **** ! Invalid name
S: CELL^1X CELL_1X
S: CELL? CELL#
```

Now you replace the "****" with the new ICED™ cell name as follows:

```
S: CELLNAMELONGERTHAN32CHARACTERSABC ABC !Cell now valid
S: CELL^1X CELL_1X
S: CELL? CELL#
```

Layer Lines

To this utility, whitespace can be a comma, any number of blanks, or both.

The layer lines in an alias file consist of the characters 'D:', 'T:', or 'L:' followed by a Stream layer number, some whitespace, then the corresponding ICED™ layer number. You may use a text editor to replace the default values of the ICED™ layer numbers with values of your own choosing. See the information below to understand the exact syntax of these layer-mapping lines.

Layer Correspondence File Format

When you provide the name of an optional layer correspondence file for the first pass, it will be copied into the alias file so that you can use the same layer correspondences many times without editing the alias file each time.

A layer correspondence file must have a .LAY file extension. It is an ASCII file consisting of a sequence of layer lines. If you wish to prepare a layer correspondence file for input to the first pass, use any ASCII editor to create default layer correspondences using the syntax explained below. (You can copy lines from an old alias file to save time.) Any lines from the layer correspondence file with Stream layer numbers actually used in the Stream file will be copied into the alias file. Layers which do not appear in the Stream file will simply be ignored. Whether or not the layer correspondence file is used, UnStream will create layer lines in the alias file only for layers actually used in the Stream file.

Layer lines control the mapping of Stream database layer numbers into ICED™ layer numbers.

NOTE: Any data mapped to ICED™ layer 0 is ignored!

Layer lines use the following syntax:

`(D | T | L): stream_layer_number optional_stream_type_number iced_layer_number`

If the optional type number is specified, the layer mapping only applies to items with the specified data or text type.

Lines that begin with a 'D:' indicate mappings from stream data types. Lines with a 'T:' indicate mappings from stream text types. Lines with an 'L:' are shorthand for identical data type and text type mappings.

Stream data type mappings affect stream objects that have data types (e.g. boundaries (polygons) and paths (wires). They also affect boxes in the stream file only when you answer with a 'Y' to the second pass prompt "Treat BOXES as BOUNDARIES (Y or [N])?" You usually respond with a 'N' to this prompt since boxes are constructs.)

If you supply the optional data type number in a data type mapping, only stream boundaries and paths with the specified data type on the indicated stream layer number be mapped onto the given ICED™ layer number. Otherwise all boundaries and paths on the indicated stream layer will be mapped onto the given ICED™ layer number. Mappings that specify a data type number override default mappings with no data type number regardless of the order in which they appear.

For example, the layer line:

```
D: 10    5    20
```

informs UnStream to map boundaries and paths on Stream layer 10 with data type 5 onto ICED™ layer 20.

If the Stream data type is not specified, the mapping applies to all items on the layer. For example, the layer line:

```
D: 10          10
```

informs UnStream to map all boundaries and paths on Stream layer 10 onto ICED™ layer 10.

```
D: 10    5    20
D: 10          10
D: 10    6    20
```

These layer lines inform UnStream to map items on Stream layer 10 with data types 5 or 6 onto ICED™ layer 20. All other items on Stream layer 10 are to be mapped onto ICED™ layer 10.

Stream text type mappings affect stream objects that have text types rather than data types. If these text type mappings should result in identical ICED™ layer numbers as the data type mappings, you can use the shorthand L layer line to define both mappings with one line.

```
L: 10    6    20
```

This layer mapping will map to ICED™ layer 20 stream layer 10 objects with either text type 6 or data type 6.

If you need more information on the use of data types and text types, see the comments in the alias file generated by the first pass of UnStream.

Stream File Translation Once the Alias File is Created

If you are using a text editor in a separate window, it is easy to forget to save the file before re-executing the utility.

After editing and **saving** the alias file generated in the first pass of UnStream, you are ready to execute the second pass.

The second pass of UnStream performs the translation. It creates a cell file for every structure in the Stream file. These cell files are placed in your current directory, even if the Stream file is stored elsewhere. If there are existing cell files with the same file names, the existing files will be destroyed. For this reason, we suggest that you use UnStream in a separate working directory. After successful conversion of the Stream file, you can copy the cell files into the appropriate directories.

To start the second pass, repeat the UnStream command with the same *path* and *sf_file_name* parameters in the same directory where you executed the first pass of UnStream.

When UnStream executes, it first searches for the alias file, *sf_file_name*.ALI. If it finds this file it will automatically start the second pass using the alias file.

Stream files from different sources vary, so the second pass of UnStream allows you several options about the format of your file. Most Stream files will be read properly using the defaults for each option. These defaults are indicated by square brackets, [], in the prompt. We recommend that you use the defaults unless you have run into a specific problem with your Stream file. To use each default, simply press the <ENTER> key.

The options UnStream will prompt you for are described below.

Enter ICED user unit size in microns [1.0]:

Reply with the size of ICED™ user units in microns. The default is to use the Stream user unit size in microns. If you want ICED™ user units in the cell files to be in mils, you would enter 25.4 in response to the above prompt.

Enter divisions per unit [1000]:

The default NDIV parameter is 1000. See the **NDIV** command line parameter on page 76 for more details on database units.

UnStream will ask you for the number of divisions per ICED™ user unit. Use the default unless you are sure that you know what you are doing. (If you do not use the default of 1000, you must specify the same number in the NDIV parameter on the ICED.EXE command line when you launch the editor to open these cell files.)

$$1 \text{ database unit} = 1 \text{ user unit} / \text{NDIV}$$

The database unit is the smallest unit of measure that ICED™ can use when storing coordinates. For example, if user units are microns and divisions per user unit is set to 1000, then all coordinates in a cell file must be aligned to a 0.001 micron grid.

The default answer to the above prompt is the number of Stream database units in 1 ICED™ user unit. For Stream file data, we recommend using 1000 divisions per unit.

Are you using UNSTREAM to scale this design or snap it to grid (Y or [N]):

The Classroom
Tutorials
manual
contains an
example of the
scaling process
in the tutorial
on Stream files.

You can change the scale of the design as it is imported by responding with a <Y> to the above prompt. If you do, then you will be presented with the following options to control the scaling process. (If not, the next prompt will be about the font height shown on the next page.)

Enter the original feature size [1.0]:

Enter the final feature size [1.0]:

The scale factor used to translate all coordinates will be equal to:

$$\frac{\text{final feature size}}{\text{original feature size}}$$

Enter snap grid size in microns [0.001]:

Your response to the prompt above defines the new grid resolution for the design. Scaled coordinates will be snapped the new grid. The new grid resolution should usually be finer than the original grid resolution or most of the scaled coordinates will be off-grid. (Even when you use a very fine grid, many coordinates are likely to be created off-grid unless the inverse of the scale factor is an even integer. E.G. the scale factor is ½.) Either use a grid resolution fine enough so that the scaled coordinates will all be created on grid without rounding, or use the parameter supplied in the next prompt to control what is done with coordinates that cannot be snapped exactly to the new grid.

Enter maximum allowable coordinate rounding error in microns [0.0]:

UnStream will translate Stream coordinates into the nearest snap grid point provided the sum of the |x| and |y| errors are no greater than the rounding error supplied in your response to the prompt above. Components failing the rounding error test will not be added to the cell file. Instead, they will be listed in the error file.

When you use a non-zero rounding error tolerance some components may be altered in a way that makes the components wider or narrower that they would have been if their coordinates were not snapped to the course grid. Another, more significant, distortion can occur in nested cells. Suppose that a deeply

nested subcell has a component with a coordinate shifted slightly to the right to be on grid. This subcell is a component in a higher level cell. Suppose that the location of the subcell in the higher level cell is also shifted slightly to the right. If each cell in the hierarchy is shifted slightly to the right, then each tiny displacement adds to the next, and the final location of the component could be significantly shifted relative to components in other cells.

This type of problem can result in an open circuit. Two shapes in different subcells may no longer overlap as they used to do in the original design. It is rare for this type of distortion to add up to much more than a single step in the resolution grid, but larger distortions can occur (and they have.)

You may want to run the second pass of UnStream twice. The first time, specify a maximum error of 0 to obtain a list of off-grid components in the *sf_file_name*.ERR file. Rounding of cell origin coordinates should be looked at particularly closely. Then run the second pass again with a large maximum error to translate as many components as possible.

Font[*n*] is "*font_name*" -- Enter character height in microns [1.0]:

The size of Stream text components is determined by the default height of a particular font multiplied by a factor stored with each text component. There is no way to extract the default height of the character font from the Stream file. Therefore, UnStream will ask for the height of each font used in the Stream file. The default value for each font is 1 micron. Please note that this is the default height of the font, not the height of the actual text components.

Use CALMA outform 3 array format ([Y] or N)?

Some newer programs no longer use the CALMA outform 3 array format. (CADENCE³⁶ software is a prime example.) It is impossible to reliably determine from the data which format the file is using. UnStream attempts to determine the format from the revision number in the file header, and sets the default in the prompt accordingly. You should use this default unless you are sure that your Stream file uses the other format. If you receive the following error messages:

³⁶ CALMA and CADENCE are registered trademarks of Cadence Design Systems, Inc.

***** HAZARD: Negative array step",
or

**** ELEMENT LOST: Unconventional or improper array step",

or if the arrays in the resulting ICED™ cell are obviously incorrect, use UnStream again with the other choice.

Convert flush end (type 0) wires to extended end (type 2) wires (Y or [N])?

The option above is included for compatibility with older versions of ICED™. You should select the default option [N] unless you are absolutely sure that you want to alter your wire definitions.

Treat BOXES as BOUNDARIES (Y or [N])?

In the original Stream format, boxes were used to mark other components for special processing, not for maskable data. However, some sources of Stream files use boxes for maskable data. If you reply with a <N> to this prompt, you will receive warnings if there are boxes in the Stream file and no equivalent components will be created for the boxes. If you reply with a <Y>, you will receive no warnings, and the boxes will be translated to maskable box components. You should always run your Stream file through the first time with a <N> in response to this prompt so you are aware if there are boxes in your file.

Ignore ELEMENT FLAGS (Y or [N])?

As of this printing, ICED™ does not process element flags. If you reply with a <N> to this prompt, UnStream will issue a warning that element flags are being ignored if it encounters any in the Stream file. If you reply with a <Y>, the warning messages will be suppressed. The element flags are ignored in either case.

After the second pass is complete, look carefully at the error file, *sf_file_name.ERR*, then use ICED™ to examine the resulting cell files. **Be sure that many false error messages in the error file are not hiding a real error message. Just one missing shape may have catastrophic results.** Only after looking very closely at the error file should you copy the cell files to your working directory.

Incompatibilities Resulting in Error Messages or Warnings

ICED™ cannot represent all possible Stream database constructs. If UnStream encounters any of the following constructs it prints an error message on the screen and records it in the file *sf_file_name.ERR*. **It is very important to look at this file carefully.** A serious error message may be hidden among many other error messages that you consider unimportant.

- ICED™ cannot handle magnified structures or structures rotated by angles other than multiples of 90 degrees. Stream structure placements (SREF's or AREF's) with these attributes are not translated to ICED™.

ICED™ text components can be converted to maskable polygons with the PGTEXT program. See the lesson on maskable text in the Classroom Tutorials Manual.

- ICED™ cannot handle TEXT with width. ICED™ treats all TEXT as if it had zero width. This means that maskable TEXT can only be translated to text components that do not produce maskable data.
- ICED™ cannot represent TEXT placed at angles that are not multiples of 90 degrees. TEXT rotations are rounded off to the nearest 90 degrees.
- ICED™ cannot handle absolute TEXT rotations. The absolute rotation attribute is ignored.
- ICED™ ignores Stream NODES.

Incompatibilities that do not Result in Error Messages

- ICED™ ignores Stream PROPERTIES.
- ICED™ only supports one font. All TEXT is converted to that font.

Note on Editor Settings of the Imported Cells

You can use the **LIMIT=ON** command line parameter when you launch ICED™ to view a large dense top-level cell created by UnStream. This will result in a much faster and easier to see initial display. All cells created by this utility have no environment database. For example, when a new cell created with UnStream is opened for the first time there will be no layer definitions, no grid definitions, and only a default resolution for coordinates.

For these reasons, when a cell created by UnStream is opened for the first time, ICED™ will execute the startup command file to define these types of settings. (Usually, the startup command file is executed only when you are creating a cell that does not yet exist.)

Be sure you have a startup command file defined for your project when you open your newly imported files or your cells will not have settings such as layer name definitions. Refer to page 25 to learn more about startup command files.

Appendix A: ICED™ FILES

.PLT
.MEN
.POK
.CEL
.LOG
.JOU

ICED™ File Types

ICED™ uses a variety of support files for different purposes. This table associates each file type extension ICED™ uses to a category covered in detail later on the following pages. Some extensions belong to more than one category.

<i>Extension</i>	<i>File Category</i>	<i>Extension</i>	<i>File Category</i>
.ALI	Layout Data in other formats	.JOU	Journal Files
.BAT	Batch Files	.JOX	Journal Files
.BB	Files used with the DRC, NLE, and LVS	.LAY	Layout Data in other formats
.BMP	Bitmap Graphic Files	.LOG	Journal Files
.CEL	Cell Files	.LVS	Files used with the DRC, NLE, and LVS
.CIF	Layout Data in other formats	.MEN	Menu Definition Files
.CL1	Cell Files	.NET	Files used with the DRC, NLE, and LVS
.CMD	Command Files	.NOW	Files used with the DRC, NLE, and LVS
.DAT	Fill Pattern Definition Files <i>or</i> Menu Definition Files	.PDF	Plotter Files
		.PK1	Files used with the DRC, NLE, and LVS
.DEL	Cell Files (previous versions of ICED™)	.POK	Files used with the DRC, NLE, and LVS
.DL1	Files used with the DRC, NLE, and LVS <i>or</i> Cell Files (prev. versions of ICED™)	.RL1	Files used with the DRC, NLE, and LVS
		.RLO	Files used with the DRC, NLE, and LVS
.DLO	Files used with the DRC, NLE, and LVS	.RUL	Files used with the DRC, NLE, and LVS
.DMP	Layout Data in other formats	.SF	Layout Data in other formats
.DOT	Plotter Files	.SHO	Files Created by the SHOW command
.EXE	Executable programs	.STI	Fill Pattern Definition Files
.EXT	Files used with the DRC, NLE, and LVS	.TAG	Files used with the DRC, NLE, and LVS
.JO1	Journal Files	.VEC	Plotter Files

Figure 127: ICED™ File Types

Executable Programs: .EXE

The operating system finds executable programs by looking through the directories stored in the system environment variable PATH.

Files with a .EXE extension in the ICED™ installation are compiled programs that can be executed from the console prompt or in a batch file. The layout editor program, ICED.EXE is one of these programs.

Executable programs are almost always stored in the Q:\ICWIN³⁷ directory. Other programs are stored in this directory, including the ICED™ utilities described earlier in this manual. See page 369 for details on executing these programs.

Batch Files: .BAT

The operating system's search path for batch files is the same PATH environment variable used for .EXE files.

Batch files are ASCII files consisting of system commands to control system variables and execute programs. You usually use a batch file to open the layout editor. We call this file the **project batch file**. See more information on creating batch files on pages 39 and 456

Batch files can also be used to open the editor to perform a specific task then quit without user interaction. This method requires no user interaction to perform the task, which can be executing "in the background" in a console window while you do work in other windows. Some examples of this type of batch file are covered beginning on page 449.

Cell Files: .CEL, .CL1, .DEL and .DL1

See page 98 for more information on cell files.

ICED™ layout data is stored in cell files that have the file name extensions .CEL, .CL1. (Previous versions of the program stored cells with names longer than eight characters in files with the extension .DEL. Backups of these files used the extension .DL1. You can convert these files to .CEL files with the Del2Cel utility.)

³⁷ Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

See page 45 for details on the cell file search path ICED_PATH.

The first time you create a cell, ICED™ will store the data in a file with the name *cell_name.CEL*, where *cell_name* is the name of the cell. The next time the cell is saved, ICED™ will backup the cell file before it is overwritten. The backup cell file name will be *cell_name.CL1*. You can use this cell backup if you make changes to your cell file that you regret later.

See **Journaling and Data Recovery** for more details on recovering cells from cell file backups.

When editing cells with ICED™, the cell files are not saved to disk until you terminate the editor. See page 440 for a detailed explanation of the steps taken when cell files are saved.

Cell files are not ASCII files and can be edited only with ICED™.

Journal Files: .JOU, .JO1, .JOX, and .LOG

The journal file is stored in the same directory as the root cell file.

ICED™ records every command executed in an edit session to a journal file with a .JOU extension. The full name of the file will be *cell_name.JOU*. This file is written to disk as the commands are executed. When you terminate ICED™ normally, this .JOU file is renamed to *cell_name.LOG*. If your system crashes, or you use the JOURNAL command to exit ICED™, the file will not be renamed.

See Journaling and Data Recovery in Appendix B.

The .JOU or .LOG files can be used to recover work in the event of a system crash or editing mistake. If the .JOU file is used in a standard ICED™ data recovery procedure, it will be renamed with a .JO1 extension while ICED™ processes the file. If the system crashes or runs out of disk space while the recovery is executing, this .JO1 file will remain on your disk. If the recovery is successful, it will be deleted.

If you elect not to use the .JOU file for recovery, it will be renamed with a .JOX extension. This allows you to recover the work if you change your mind after viewing the cell. The .JOX file will not be deleted automatically by ICED™, but it will be overwritten without warning.

The menu option **1:FILE→edit.JOU** can be used to view the journal file.

The journal files are written in executable command syntax. Journal files are useful to determine the exact syntax of commands generated when you use the menus to execute a command. You can edit journal files to create command files to be executed in future ICED™ sessions.

Command Files: .CMD

Use the
@file_name
command to
execute
command files.

Normally, ICED™ commands are entered from the keyboard or selected from menus. However, ICED™ commands can be typed in any ASCII text editor, saved in a file, and then executed as a unit during an edit session. Several ICED™ commands are created specifically for use in command files. The extension used for command files is usually .CMD, however, any extension is valid.

See page 98 for
details on the
command file
search path.

Command files written to be called only by other command files to perform sub-tasks should have file names that begin with an underscore ('_'). Command files with this prefix will not be included in the lists of command files generated within the editor by the *@%.cmd* menu option. Using this prefix for command files that you will never execute directly in the editor limits the clutter added to these lists and makes it easier to find the desired command file.

Files Created by the SHOW Command: .SHO

The ICED™ SHOW command can create an ASCII listing of components and macros. These files have a .SHO extension by default, although they can be created with any file extension. Since these files are in command format syntax, they can be executed as command files.

Layout Data in Other Formats: .SF, .CIF, .ALI, .LAY, and .DMP

ICED™ supports importing and exporting data from both Calma GDSII Stream Format files (.SF file extension) and Caltech Intermediate Format³⁸ files (.CIF extension).

Stream files are imported into ICED™ cell files with the UnStream utility. ICED™ can export a Stream format file with the STREAM command. The SFDmp utility can be used to create a readable dump of a Stream Format file. The files created by SFDmp have a .DMP extension.

³⁸ The CIF standard is defined in "An Introduction to VLSI Systems" by Lynn Conway and Carver Mead, Addison-Wesley Publishing Company, 1980.

CIF files are imported into ICED™ with the UnCIF utility. ICED™ exports cell data into CIF files with the CIF command.

Both the UnStream and UnCIF utilities use an intermediate alias file with a .ALI extension. Both utilities use an optional layer definition file with a .LAY extension. These files can be edited with any ASCII editor.

Bitmap Graphic Files: .BMP

The advanced batch file example on page 451 shows you how to create a bitmap graphic of a cell by clicking a shortcut on your desktop.

.BMP bitmap files store an image in a widely accepted graphics format. These files can be added to documents by most desktop publishing programs (e.g. Microsoft WORD) or sent to a printer directly by any graphics utility (e.g. Microsoft Paintbrush).

These files are created by the MkPlot utility from data exported by the PLOT command. Refer to those descriptions for more information. The complete list of steps to create a bitmap graphic file is shown on page 281.

Plotter Files: .PDF, .VEC, .DOT, and .PLT

See page 44 for the correct location of auxiliary files such as .PDF files.

The **MkPDF** utility can display or modify information in a .PDF file.

The file ICED™ uses to control plot output for a specific plotting or printing device is a Plotter Definition File with a .PDF extension. ICED™ supplies many of these files to control plotting on a wide variety of printers and plotters. These .PDF files are used by the PLOT command and the MkPlot utility.

The files generated by the PLOT command have a .VEC extension. The MkPlot utility either sends the data in the .VEC file directly to a printer or plotter, or creates a file with a .DOT extension that you must send to a printer explicitly. See the MkPlot utility description for examples.

None of these files are ASCII files, so they can not be edited.

Fill Pattern Definition Files: .DAT and .STI

See page 44 for the correct location of auxiliary files like .STI files.

ICED™ provides you with the ability to design your own stipple patterns for filling component outlines on plots and for screen display. The MkSti utility is used to create stipple pattern files. The files used for input to MkSti have a .DAT extension. (This is the same extension used by the MkMenu utility, although the format of the data is not the same.) The files created by the MkSti utility (and used by the PATTERN and PLOT commands) have a .STI extension. The .DAT files are ASCII files and can be created by any ASCII text editor. The .STI files are compiled and built for speed, and as such they cannot be browsed or edited.

Menu Definition Files: .DAT and .MEN

.MEN files should be located on the auxiliary file search path. See page 44.

You can design your own menus for ICED™. The files used to define the menu structure have a .DAT extension (which is the same extension as files used to define fill patterns). See the sample ASCII menu definition file Q:\ICWIN\SAMPLES\M1.DAT. These files are created with any text editor, then processed by the MkMenu utility to create files with a .MEN extension. Files with a .MEN extension should not be edited.

Files Used with the DRC, NLE and LVS Utilities: .BB, .DLO, .DL1, .EXT, .LVS, .NET, .NOW, .POK, .PK1, .RUL, RLO, .RL1, and .TAG

See the Internal DRC tutorial in the Classroom Tutorials Manual to learn about executing the DRC from the ICED™ menus.

The DRC program (Design Rules Checker) uses a variety of file types. Input rules files have extensions of .RUL. These files are compiled to files with a .BB extension. These files define layout specifications used to verify and manipulate design data. The DRC command may need to refer to one of these files to determine if extra design data should be included to verify shapes in the border area of your selected design data.

The NLE (Net List Extractor) utility uses similar rules files to extract device lists from the design data. The LVS (Layout Vs Schematic) utility compares the device list from the layout data with your schematic data.

Refer to the manuals for these utilities for more details.

One File That Should Not Be Deleted Or Copied

The ICED3AUX.EXE (or ICED5AUX.EXE file in newer installations) file controls ICED™ copy protection. It is customized to match your hardware key. If you include it in a list of files sent to another ICED™ user, you will overwrite their customized copy. Their ICED™ installation will not function until the overwritten file is restored from their original media.

Appendix B: Journaling and Data Recovery

“OOPS !!!”



How Cell Files are Saved

Cell files are saved when the layout editor terminates. See the EXIT command description to learn which termination commands flag a cell for saving and which do not.

The complete series of steps for saving each cell file is listed below. If one of these steps fails due to some system problem like a lack a disk space, the layout editor issues a "SAVE ERROR" message, skips the remaining steps for that cell, and goes on to the next cell. (The error messages are displayed on the screen and recorded in the journal file.) It is rare for problems to occur during these steps, but if you get a warning message at this stage:

Do not try to recover using the journal file and automatic recovery. Do not launch the layout editor until you have copied all files to a backup and renamed the affected files.

Instead, you should complete the failed/skipped steps by hand. No data has been lost, but rash actions at this point can lose data.

When you exit the layout editor, it goes through the following steps for each cell:

- The cell file is written to the file *cell_name*.TMP.
- If both cell files *cell_name*.CEL and *cell_name*.CL1 already exist, *cell_name*.CL1 is deleted.
- If the file *cell_name*.CEL exists, it is renamed *cell_name*.CL1.
- The file *cell_name*.TMP is renamed *cell_name*.CEL.

After these steps have been completed for all cells ICED™ renames the session's journal file from *root_cell_name*.JOU to *root_cell_name*.LOG.

The Journal File

See page 99 for an overview of journal files.

The journal file is the key to the recovery mechanism of ICED™. During every ICED™ session, every command executed is also echoed in the journal file. This is true whether commands are typed in or selected from the menus. If for any reason the cell is not saved at the end of an ICED™ session, all your work can be recovered by executing the commands in the journal file. This allows you to recover from system crashes.

The journal file can be modified using any ASCII text editor. The commands in the file can be modified to correct mistakes before using it for recovery. In this manner, the modified journal file can be used to recover from editing mistakes that UNDO cannot reverse (e.g. discovering that you deleted the wrong group of components after executing several more edit commands).

The menu option **1:FILE→edit.JOU** can be used to view the journal file.

The journal file is created as soon as you launch ICED™. The name of the file will be *cell_name.JOU*, where *cell_name* is the name of the cell you launched ICED™ to edit. The journal file will be located in the same directory as the root cell file. The journal file is updated as you execute each command. When you terminate ICED™ normally, the file is renamed *cell_name.LOG*. If an old file exists with this name, it is deleted.

If your system crashes or you terminate ICED™ with the JOURNAL command, the journal file is not renamed. (Under these circumstances, no cell files are saved. Cell files are saved to disk only when ICED™ is terminated normally.)

Journaling During Command Files

The LOG command also controls how comments are generated in the journal file.

When executing a command file, you have the choice of logging just the command that calls the command file, or logging each command in the file. It is recommended to log each command in the command file. (This is due to the fact that recovery of the data is guaranteed if all commands are logged into the journal file. If only the command that called the command file is logged in to the journal file and you edit the command file before recovering, the edited command file will be executed instead of the original command file.) Logging each command in the command file is the default unless a LOG OFF command is used as the first command in a command file.

If the LOG mode is not turned off, the commands in the command file are stored in a buffer and dumped into the journal file as the buffer fills. (This is much faster than opening and closing the file for each command.) When the command file is completed, or when an error is encountered, any commands remaining in the buffer are dumped into the file.

See **Recovering From Mistakes** on page 444 for more important information on editing the journal file before using it for recovery.

If your system crashes during the command file, some commands may not be written to the file. If you use the journal file for recovery, you should always remove all commands from the end of the file up to the command that called the command file. It is important to execute an entire command file as a unit.

Automatic Data Recovery

When you launch ICED™, one of the first tasks it performs is looking for a file with the name *cell_name*.JOU. If one exists, you are given the option of performing automatic data recovery. (A file with the name *cell_name*.JO1 will also trigger automatic recovery. .JO1 files are explained below).

Let us say that you are in ICED™ editing a cell with the name OPAMP. The power to your system is then cut off. The cell file OPAMP.CEL was not saved, and the file OPAMP.JOU was not renamed to OPAMP.LOG. When your power is restored and you launch ICED™ again to edit **the same root cell**, ICED™ recognizes automatically that edits to OPAMP were not saved and displays a message similar to this:

**Journal file for Q:\ICWIN\CELLS\OPAMP exists
Do you want to recover [YES, NO, or QUIT]**

If you type <N> at this point, the journal file is renamed to OPAMP.JOX and no work from the last ICED™ session is repeated. (You can still use the .JOX file for recovery later if you change your mind after viewing the cell file. Simply execute the command @OPAMP.JOX from inside ICED™.)

If you type <Y> at this point, ICED™ will use the commands in OPAMP.JOU to recover all of your work. The cell will be updated to the state it was before the power was cut off.

This process is automatic. You do not even have to remember that your work was interrupted. If cells other than OPAMP were modified during the same edit session and not saved by ICED™, they will all be recovered automatically.

Suppose that something goes wrong while ICED™ is performing the recovery. If your system crashes or runs out of hard disk space while ICED™ is processing the journal file, you are still protected.

Before ICED™ starts the recovery process, it renames the *cell_name*.JOU file to *cell_name*.JO1. As ICED™ processes the journal file, it will log all commands as they execute into a new *cell_name*.JOU file. Once the recovery is complete, ICED™ will delete the .JO1 file. However, if the recovery is interrupted, the .JO1 file will still exist. ICED™ detects this when it is launched the next time and will begin the recovery from the beginning again using the .JO1 file.

Recovering From Mistakes

If your system encounters problems while ICED™ is actually writing the cell files, see page 440 for details on recovering the files.

The **JOURNAL** command terminates the editor without saving any cell files.

The journal file is simply a list of commands that modified your cell. You can edit these commands before you use the journal file for recovery to recover the cell at a state it was in before a mistake was made.

Suppose that you are in ICED™ editing a cell. You use the DELETE command to delete a selection of components and do not realize that it was the wrong group of components until later on in the same session. It is too late to use the UNDO command, since the UNDO command only reverses the last edit command executed. At this point, you can use the JOURNAL command and the journal file to recover only the commands executed prior to the unfortunate DELETE command.

We strongly recommend that you backup the cell files and journal file before you begin this type of recovery.

Use the JOURNAL command to terminate ICED™. No cell files are saved, and the *cell_name*.JOU file is not renamed to *cell_name*.LOG. You can now edit the *cell_name*.JOU journal file with any ASCII text editor.

The Classroom Tutorial Manual contains several lessons with examples of this process.

Starting at the end of the journal file, search upwards for the DELETE command that caused all the trouble. Remove the line of the journal file with the DELETE command **and all lines following the DELETE command down to the end of the file**. (It is important to realize that the state of a cell depends on each command which modifies it. Trying to remove only one command from a journal file and then executing succeeding commands will often lead to unexpected results. The cell is no longer the same cell it was the first time those commands were executed.)

When you have saved the modified journal file (do not change the file name), launch ICED™ as usual and it will recognize that data recovery should be performed because the file *cell_name*.JOU exists. The "Do you want to recover?" message will be displayed. Type a <Y> at the prompt and your cell will be restored to the state it was before the DELETE command was executed.

Recovery After Cell Files Are Saved

Even once a cell file is saved, it is still possible to recover from errors made during the last ICED™ session.

Before ICED™ overwrites an existing cell file, a backup copy of the cell file is made to the file *cell_name.CL1*. If you use the EXIT or LEAVE command to end editing a cell, and a cell file is saved with errors when you terminate ICED™, you can use the cell file backup to recover the cell as it was before the edit session began. If you want to recover part of the work done during the last edit session, you can use the journal file.

If you wish to recover a single cell as it was before the last ICED™ session, simply rename the cell file (or delete it) and copy the cell file backup to the appropriate cell file name.

Example:

```
REN NAND.CEL NAND.BAD  
COPY NAND.CL1 NAND.CEL
```

When these commands are typed at the console prompt, in the directory where the cell you wish to recover is stored, they will replace the cell file with the version saved before the last edit session.

If you modified more than one cell during your last session, you can use the list at the end of the journal file to see which cell files were saved during that session.

Remember that the journal file is renamed to *cell_name.LOG* when you terminate ICED™ normally.

If you realize that you corrupted a cell only after viewing it in a new ICED™ session, you must be careful in terminating that session, so you do not overwrite the cell file backups and journal file. Use the **JOURNAL** command to terminate the session, then delete the new *cell_name.JOU* file so automatic recovery is not performed the next time you edit the cell.

Let us take a fairly complicated example. Let us suppose that you modified two cells during your last ICED™ session. You used the EXIT command to terminate editing the cells and both cell files were saved. The next time you use ICED™ to edit the cells, you see that some pretty serious errors were made, and you want to recover the cells as they were before you began the previous edit session.

When you terminate the editor with the EXIT, LEAVE or QUIT commands, any old file with the name *cell_name*.LOG is deleted.

You should use the JOURNAL command to terminate the current ICED™ session so the backup cell files and journal file from the previous session (*cell_name*.LOG) are not overwritten. Before launching the editor again, be sure to rename or delete the new *cell_name*.JOU file. The presence of this file would cause ICED™ to assume that you want to perform automatic recovery of the edits made in the session you just ended, and that is not the case. If you do not delete the *cell_name*.JOU file, be sure to reply with a <N> to the automatic recovery prompt.

The journal file from the edit session where you made the errors is stored as *cell_name*.LOG. You should look at the end of this file for the list of cells saved during that session. The list will look something like this:

```
! Saving E:\ICWIN\CMOS\NAND.CEL
! Saving E:\ICWIN\CMOS\TEST.CEL
```

Now replace all cells saved with errors during that session with the cell backups from the previous session. You can use the following DOS commands in the console window, or rename the files with the file manager of your choice:

Example:

```
REN NAND.CEL NAND.BAD
REN TEST.CEL TEST.BAD
COPY NAND.CL1 NAND.CEL
COPY TEST.CL1 TEST.CEL
```

You can also recover part of the work done during the session where you made the errors. You edit the *cell_name*.LOG journal file to remove the commands executed in error and all commands after them. You then execute the modified journal file on the cell file backups. **Read the rest of this section carefully before you start.**

We strongly recommend that you perform this type of recovery in a temporary working directory rather than in your design directories. Overwrite your old cell files with the recovered cell files only after you have made sure the process went as you expected.

If the cell you are recovering has cells nested inside of it, the nested cells which are stored in the same directory as the cell you are recovering must be copied to the temporary directory for you to open the cell. The easiest way to do this, is to copy the entire directory containing the cell to be recovered to the new temporary directory.

If you edited more than one cell during the ICED™ session where you made the errors, you **must** recover all cells saved during that session. Make sure to copy all backups of cells modified during that session to the new directory. The list of cells saved during that session is recorded at the end of the journal file. The cells might be stored in other directories. **Make sure that the backup cell files of all affected cells are in the temporary directory.**

While in the new directory, you must rename the cell files of the cells you are about to recover to something like *cell_name*.BAD. Then copy the cell backup file(s) to proper cell file names as shown in the example above. Copy all *cell_name*.CL1 files to *cell_name*.CEL.

Now edit *cell_name*.LOG to remove the commands which were executed in error **and all other commands which follow until the end of the file.**

Using the new directory as your working directory, launch ICED™ to edit cell *cell_name*. (No recovery option will be offered because no .JOU file exists.) The cell will now be in the state it was before the edit session with the mistakes began. Execute the command *@cell_name*.LOG to execute all the changes made to the cell(s) during your last edit session before the errors were made. If the process went smoothly, EXIT from all cells. If the recovered cells look as you expected, overwrite the cells in your original working directory with the recovered cells.

Appendix C: Advanced Batch File Examples

The two examples that follow open the layout editor and then automatically execute a command file using the QUIT command line option. This method allows you to repeat the exact same procedure each time without the need for user interaction. This method is very desirable for many layout activities, particularly for exporting design data for design verification (the DRC or LVS programs) or for final design export for fabrication. You may want to use this method with critical tasks that must be performed exactly the same way each time (e.g. stream file export.)

The first example calls the project batch file so that the latest project settings are used automatically. With this method, you don't have to worry about editing your re-usable batch file when the project changes (e.g. a change in the cell libraries). This example plots a cell when you click a button on your desktop.

If you prefer to disconnect your task from the project so that changes to the project definition cannot affect how it executes, see the example on page 456. This example uses the project batch file as a template to create a new batch file that exports information for the DRC (Design Rules Checking.)

You should read the information in the "**Project Batch Files**" chapter beginning on page 39 for a general overview on using a batch file to launch the layout editor before modifying either of these examples for your own uses.

Creating a Desktop Shortcut to Plot a Cell

It is best to execute this type of batch task when the layout editor is not currently open.

The example below creates a shortcut on your desktop that will create a bitmap file of the current layout of a cell with a single click. With simple alterations, the same steps can be used to plot a cell on your printer or plotter.

The PLOT command in the layout editor creates only an intermediate export file. To actually create the bitmap file (or plot the data) an additional utility must be executed. This procedure can be somewhat cumbersome, but when you automate the task with a batch file it can be accomplished with a single mouse click.

See the next example on page 456 to create a batch file independent of the project batch file

This method uses the project batch file so that current project settings are always used without requiring edits to the PLOT.BAT batch file to keep it up-to-date when the project changes (e.g. cell library path changes).

For this example, we will assume that you have already created a project batch file MYPROJ.BAT . We will assume that the working directory you are using is Q:\ICWIN\WORK.

Creating the PLOTIT Command File

Learn more about command files on page 98.

Open the layout editor normally with your project batch file. Execute the PLOT command with the menu options FILE->PLOT. This interactive method of executing the command allows you to select the appropriate options easily. However, it is easier to repeat this procedure by re-executing exactly the same command every time you need to create a new plot.

Refer to the **PLOT** command description for details on using these options.

- Select the "BMPCOLOR" plotting device to create a bitmap file. (Select your plotting device if you prefer to create a task that plots the data on your plotter, however some of the options presented may vary from those shown below.)
- Next select the pattern file. Use "SAMPLE" unless you use a different file.
- For Super Pixel Mode, select "1 Dot".
- For Size, select "AA".
- Finally select "None" to avoid further options. This will plot the entire cell at a scale that fits the selected size.

If the command is successful, you will see a message similar to:

Output written to file Q:\ICWIN\WORK\MYCELL.VEC

(If your plot command is not successful, read the PLOT command description to determine the cause of the problem and take whatever steps are required to execute a successful PLOT command.)

Learn about the journal file on page 99.

The command that created the plot file has been echoed in your journal file. We will edit this file to extract the syntax of the command so it can be easily repeated.

Close the layout editor with the QUIT command.

The menu option **1:FILE→edit.JOU** can be used to edit the journal file.

Edit the journal file, Q:\ICWIN\WORK\MYCELL.LOG, with your favorite text editor. Delete all lines from the file except for the successful PLOT command and the QUIT command at the bottom of the file. The PLOT command will probably be longer than a single line, be sure to select all of it. The lines should look similar to the following:

Example:

```
PLOT PLOTTER="BMPCOLOR" PATTERN="SAMPLE" SIZE=AA &  
ARRAY_MODE=FULL DOTS=1 TEXT_ORIENTATIONS=2 &  
TEXT_HIGHLIGHTS=DISABLED CELL_LABELS=ON &  
STRIP=YES R0 ENGLISH ALL=51.037073
```

The QUIT command will make the editor terminate after executing the PLOT command:

Now **save the file with the new name "PLOTIT.CMD"** in the same directory.

Create the PLOT Batch File

Open your favorite text editor to create the batch file. This file should also be located in the same working directory. We will use the file name PLOT.BAT. Type the following lines:

Example:

```
CALL Q:\ICWIN\39MYPROJ.BAT
Q:\ICWIN\ICED.EXE %1 MAX=NO PAUSE=0 QUIT=PLOTIT.CMD
Q:\ICWIN\MKPLOT %1 /R:60
```

Refer to page 39 to learn more about project batch files and to see an example file.

The first command calls the project batch file. You must use a CALL command to do this so that control returns to the other commands in the PLOT.BAT file when the project batch file completes. Since no cell specification is used in the command line of the project batch file, only the lines in this file that define system environment variables are executed. The batch file then terminates and control is passed to the next line in the PLOT.BAT file.

The layout editor command line comes next. The %1 takes the place of the cell name so that we can pass the name of the cell we wish to plot as an argument to the PLOT.BAT batch file.

See the BATCH option on page 60 for details on using # in batch file arguments in cases where = is stripped from options by the operating system.

The rest of the ICED.EXE command line options result in overriding the maximize mode, the pause mode, and the startup command file. The result will be that the editor opens briefly to execute the PLOTIT.CMD command file. Since the command file is executed with the QUIT command line option, the editor terminates with a QUIT command.

³⁹ Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

Refer to the **MkPlot** utility to learn more about bitmap resolutions.

The next line of the PLOT.BAT file shown above calls the MkPlot utility. The call name will take the place of the "%1" when the file is executed. The "/R:60" parameter is used to set the resolution of the bitmap file. You can use whatever value you require. (If you are revising this example to plot to a plotter, you may not need this parameter.)

Save the file with the name PLOT.BAT in the working directory.

Create the Desktop Shortcut

See page 90 for other methods of executing batch files in Windows.

Once you have created the two files described above, it is time to create the shortcut that will execute them with a single click from your desktop. The easiest way to do this is from Windows File Explorer, usually opened with the "My Computer" desktop icon.

Navigate to the Q:\ICWIN\⁴⁰WORK working directory. Click the left mouse button once to select the PLOT.BAT file. Now press down and hold the right mouse button while you drag the cursor to an empty spot on your desktop. Let the mouse button up. On the popup menu, select "Make Shortcut(s) Here."

Our last step is to modify the properties of this shortcut. Click on the new shortcut with the right mouse button. From the pop-up menu, select "Properties." On the new dialog, select the "Program" tab. The name of the batch file should already be shown on the "Cmd line" field. Add the name of the cell you wish to plot after the name of the batch file. The text you type after the name of the batch file will replace the string "%1" in the batch file commands. The "Cmd line" field should now look similar to the following:

Example:

Q:\ICWIN\WORK\PLOT.BAT MYCELL

The "Working" field should be set to the working directory already.

In the "Run" field, you should use the down arrow to select "Minimized".

⁴⁰ Remember that Q:\ICWIN represents the drive letter and path where you have installed ICED™.

Use the mouse to check the "Close on exit" box.

Click OK to close the dialog.

Plotting with the Shortcut

To create the bitmap file MYCELL.BMP all you need to do is click on the shortcut icon with the left mouse button. You'll know the task is complete when the new button disappears from the taskbar.

If something goes wrong and the MkPlot utility obviously uses the wrong data or pauses waiting for input, it can be cancelled by typing the <Cntrl><C> keyboard combination.

To create a bitmap of a different cell, simply modify the shortcut properties. (Right click the shortcut icon, select "Properties", and click the "Program tab.") Replace the cell name in the "Cmd line" field with any cell in the working directory and click OK. Execute the shortcut by left clicking it.

Creating Batch Files for Repeatable Procedures

The project batch file can be used as a template for creating batch files to perform many layout tasks. Since it already defines the cell libraries and other project settings, it is easily customized to open the layout editor and to perform a specific task. However, unlike the previous example, the method below disconnects the new batch file from the project settings, and changes to the project batch file will not be reflected automatically in your new batch file.

This can be desirable in some situations. Accidental changes to the project settings in your project batch file (e.g. temporarily changing the order of the cell libraries) will not affect how your batch file executes. A "disconnected" batch file (one that does not use the current project batch file) will perform exactly the same procedure using the exact same cell libraries each time.

See page 39 for more details on project batch files.

We will modify a project batch file and copy it to a new name in this example. A project batch file defaults to opening the layout editor interactively. A few simple alterations are required to open the layout editor to execute a few specific commands and then quit.

Using the layout editor as a batch procedure, rather than interactively, insures that a task is performed exactly the same way each time. This makes it ideal for critical tasks like stream file export or final design verification. This example will export your design to a Stream file suitable for submission to a foundry and export a file for DRC (Design Rules Checker) or LVS (Layout Vs. Schematic) design verification and at the same time. This insures that the layout sent to foundry is the same layout validated with the verification tools.

Modifying the Project Batch File

In the following example we will copy and modify a project batch file to automate the export process. Copy the project batch file shown on page 40 if you do not have a project batch file of your own.

Copy the file to a convenient directory and rename it DRCEXPORT.BAT. Open this file with your favorite text editor.

If you do not locate the batch file in the working directory where your top-level cell is located, add statements similar to the following to the top of the file to select the working directory and its drive letter.

Example:

Q:⁴¹
CD \ICWIN\WORK

Leave the SET commands that create the environment intact. Remove the conditional statements (and the :end" statement). You need to modify the statement that launches the layout editor. Before modification this line should look similar to the following:

Q:\ICWIN⁴¹\ICED %1 START=... MENU=* PAUSE=0 ...

Create a windows shortcut to the batch file to close it on exit or pass arguments on the command line. See page 454.

The %1 parameter is a placeholder that is replaced with the first argument on the command line of the batch file. This parameter specifies the name of the cell to open. You can replace this with your top-level cell name if desired, then you will not have to pass a cell name in the argument list of the DRCEXPORT.BAT command line.

For this example, we will not support argument passing. Replace the %1 with your top-level cell name. Replace the other parameters so that your command line looks similar to:

Example:

Q:\ICWIN\ICED MYCELL QUIT=DRCEXPORT.CMD PAUSE=0

⁴¹ Remember that Q: represents the drive letter where you have installed ICED™. ICWIN represents the path defined during installation. Replace these with your installation drive letter and path.

The command line above will launch the layout editor and the commands in the file DRCEXPORT.CMD will be executed. Since the command file is executed with the QUIT command line option, the editor will terminate without overwriting any cell files and without the need for any user interaction.

Create the DRCEXPORT Command File

See page 98 for more details on command files.

A command file is simply a list of layout editor commands. The DRC command creates the data file for the DRC (Design Rules Checker) or NLE (Net List Extractor) programs. The command required to export the DRC data file is as simple as the following:

Example: **DRC** !creates DRC/NLE export file

The DRC and Stream commands are fully described in the command reference section of this manual.

The commands for Stream file export are only slightly more complicated. You will need to add a command that executes a technology dependant file to assign stream numbers to every exportable layer. This is often done with the startup command file. Stream numbers are deliberately not saved with cell files. (You cannot use the @* command to execute the startup command file, since @* will just execute the file specified by the QUIT command line parameter.)

Example: @*startup_cmd_file_name* !replace with the name of your startup command file
STREAM MAP !creates stream file

Place these three lines (and any relevant comments) in a text file and name the file DRCEXPORT.CMD. It is best to locate it in your technology directory Q:\ICWIN\TECH\mytech\. This should insure that it will be found by the editor without specifying the path. If it is not in this directory, add the path to the file name in the QUIT parameter in the DRCEXPORT.BAT file.

Executing the DRCEXPORT.BAT File

A batch file can be executed with the methods described beginning on page 90.

The easiest way is to double click on the file from Windows File Explorer. (You may need to close the console window opened for the batch file when it is finished by clicking on the button with the X in the upper right corner of the window.)

Index

- !
- allowing in Stream structure name 409
- !comments 121
- generated by commands 238
 - in command files 34
- #
- allowed in cell names 96
 - in UnStreamed cell names 420
 - prefix for macro names 232
 - using instead of = in batch file 60
- \$
- allowed in cell names 96
- \$\$comments 121, 237
- \$comment command 113, 120
- example 360
 - in startup cmd file 34
- %
- using for macro reference 120
 - using in batch file 49
- % parameter placeholder 53, 60, 142, 221
- &
- obsolete batch continuation char 44
- ()'s
- used to indicate required parameters 102
- *****
- in UnStream alias file 421
- ...
- used to indicate continuation 105
- ?
- in Stream structure names 420
 - in UnStreamed cell names 421
- @%cmd menu option 99, 117
- @*
- use only w/STARTUP on cmd line 458
- @* command 116. *See @file_name* command
- using for export initialization 32
- @.VM, @.TMP, and @@.TMP 84
- @file_name command 113, 115
- example in another cmd file 381
- @file_name command line parameter 56
- @string.DEL cell name format 97
- []'s
- used to indicate optional keywords 103
- ^
- in Stream structure names 421
-
- allowed in cell names 96
- _LOOP.CMD command file 38
- +
- using to split ICED_HOME definition 44
 - using to split ICED_PATH definition 47

Index

- <>'s used to indicate keystroke 105
- <Esc> key 126
- =
 - how to avoid stripping in batch file 60
- > DOS redirection operator 386
- Aborting a command file 361
- Aborting utilities 377
- ADD command 122
 - ADD ARC 137
 - ADD ARRAY 148
 - ADD BOX 129
 - ADD CELL 146
 - ADD CIRCLE 132
 - ADD LINE 140
 - ADD POLYGON 129
 - ADD RING 133
 - ADD SECTOR 140
 - ADD TEXT 141
 - ADD WIRE 134
 - AT keyword optional 125
 - changing window during 126
 - creating with SHOW command 308
 - defaults 123, 216, 346, 351
 - defining positions 125
 - errors during command files 132
 - keyword order 124
 - LAYER keyword 124
 - OFFSET keyword 127
 - overview of related commands 110
 - setting default layer 216, 351
 - setting default N_SIDES 352
 - using spacing cursor 322
- Alias file
 - UnCIF32 412
 - UnStream 420
- Aligning components 253
 - spacing cursor 322
- AllCells 378
- AllCells utility 38
- Always command files 35
 - auto-executing on open 58
 - executing with @* command 116
- ALWAYS command line parameter 453
 - alternative methods 91
 - use of always command file 35
- Angle
 - snap 320
- Angles
 - calculations of arc components 139
- Applications
 - executing from editor command line 329
- Arc components
 - angles 137, 139
 - creating 137
 - setting default N_SIDES 352
 - setting default type 352
 - use of N_SIDES 139
 - vertex calculation 127
 - wire type 137
- Archiving cell files 333, 334
- Area
 - maximum definable 76
 - reporting component area 313
- ARRAY command 112, 151
 - in startup command file 34
- Array components
 - adding/creating 148
 - arrays of arrays 148
 - defined 94
 - depth defined 182
 - DRC output of 188
 - example 149
 - mirroring 149
 - one-dimensional 148
 - origin 149, 150
 - outlines 162
 - outlines, displaying 256
 - plotting 278
 - plotting outlines 271
 - rotating 149
 - selecting 298, 303

-
- speeding screen refresh151
 - step148
 - Stream file translation427
 - using bounding box to select300
 - using layer 0 to select297
 - ARROW command112, 153
 - in startup command file34
 - Arrow keys
 - panning vs. history153
 - AUTOEXEC.BAT373
 - Automating layout tasks91
 - batch files450
 - Autopan156
 - AUTOPAN command112
 - in startup command file33
 - AUXIL directory17, 21
 - searched for command files48
 - Auxiliary files16
 - Aztech directory17
 - Background of editor window29
 - Backslash
 - allowing in Stream structure name409
 - use in map file409
 - BATCH command line option60, 91
 - Batch files14
 - assigning to file extension90
 - AUTOEXEC.BAT373
 - automating procedures91
 - calling one batch file from another376
 - DRCEXPORT.BAT457
 - executing from desktop shortcut454
 - executing from Windows Explorer376
 - executing in Windows90
 - executing nested file with CALL453
 - interaction-free example456
 - passing arguments60
 - passing arguments53
 - PLOT.BAT453
 - referring to environment variables49
 - suppressing console messages60
 - using editor in batch mode450
 - using to launch layout editor40
 - utility to visit all cells378
 - XCELLS.BAT378
 - Batch jobs
 - avoiding user interaction ... 58, 65, 69, 79, 451
 - BEEP command line parameter62
 - Bitmaps281, 451
 - BLANK command111, 112, 158
 - Blank spaces
 - invalid in cell names96
 - Blank view window
 - refresh289
 - Blanked components
 - where status is stored95
 - Blanking components158
 - Blanks
 - in Stream file structures409
 - in Stream structure names421
 - BLINK command112, 163
 - BMPCOLOR281
 - BMPCOLOR plotting device452
 - BMPMONO281
 - Boundaries
 - assigning pen # for plot . 222. *See also* Layer 0
 - plotting outlines271
 - Boundaries of cells112
 - Bounding boxes147, 256
 - turning display off183
 - Box components
 - created from polygons131
 - creating129
 - cutting180
 - merging as wires243
 - selecting298
 - selecting sides295
 - spacing cursor322
 - Boxes
 - in Stream files428
 - CALL MS-DOS command376, 453
 - Calma GDSII-Stream file*See* Stream file
 - CANCEL command line parameter63

Index

- Cancelling commands 63
- Case
 - in stream file structures 420
 - in Stream files 407, 420
 - in text components 348
- CD MS-DOS command 23, 90, 374, 376, 457
- Cell backups 13
- Cell boundaries 112
- Cell depth
 - defined 182
- Cell file search path *See also* Cell libraires
 - definition in batch file 45
- Cell files 13
 - .DEL files 378, 382
 - archiving 333, 334
 - assigning extension to ICED™ 90
 - avoiding saving 211
 - backups 98
 - blank flags saved in 159
 - cif/stream parameters not saved in 227
 - corrupting 311, 312
 - creating from CIF files 412
 - creating from Stream files 418
 - determining library 384
 - EDIT command 192
 - finding top-level 388
 - geometric vs. environment data 95, 148, 340
 - Key macros stored in 214
 - library definition in batch file 45
 - location of new cells 19
 - menu stored in 74
 - missing 97
 - protecting from changes 45, 85
 - saving 193, 198, 206, 287, 341
 - saving only when changed 229
 - when saved/not saved 100
- Cell libraries *See also* ICED_PATH
 - changing layer numbers w/cmd file 380
 - defined 18
 - definition in batch file 45
 - direct edit 47
 - duplicate cell names 19
 - editing cell in 20
 - executing cmd file in each cell 378
 - missing 97
 - not searched for root cell 51
 - protection classes 45
 - read-only 46
 - reporting for nested cells 384
 - used to limit SHOW data 310
- Cell name restrictions 96
- Cell outlines 112
- Cells
 - adding arrays of 148
 - adding to design 146
 - blanking 158
 - bounding boxes 256
 - case of names in Stream files 337
 - changing layer numbers w/cmd file 380
 - creating 192, 206, 258
 - creating without user interaction 66
 - current cell name 314
 - defined 94
 - determining cell names 307
 - editing 192, 258, 340
 - editing cell in library 20
 - executing command file in each 378
 - finding top-level file 388
 - geometric vs. environment data 95, 192
 - grouping/ungrouping 206
 - labels, displaying 182
 - listing components in 310
 - listing nested cells 384
 - mirroring 147, 248
 - origin 147, 207
 - outlines, displaying 256
 - plotting outlines 271
 - plotting to specific depth 276
 - protecting from changes 285
 - rotating 147, 292
 - selecting 298
 - specifying root cell in cmd line 51
 - swapping 111
 - using layer 0 to select 297

viewing while preventing changes.....	85	data stored in cell file	148
CHANGES-LAY.TXT	378	defining	167
Changing Layer Numbers	37	line/text color varies with fill mode	141
Child cell	<i>See also</i> Subcell	listing current settings.....	344
defined.....	95	of grids	203
environment database ignored/replaced	95	overlapping.....	171
exiting.....	100	setting layer properties	31
Choosing components.....	<i>See</i> SELECT command	comma	
CIF command	113, 165	allowing in Stream structure name	409
CIF export		Command file	
automating.....	92	aborting	361
location of exported files.....	19	Stream file export	410
CIF file		Command File Programmer's Reference Manual	8
layer names in startup cmd file.....	32	Command file search path	
missing cells	97	defining in batch file.....	48
speeding first time display.....	70	Command files	
CIF files	435	\$\$comments as progress indicators	237
automating export.....	458	ADD command errors	132
blanking does not affect	158	assigning export properties	32
creating.....	165	auto-executing on open.....	58, 65, 69, 79
layer assignments not saved	165	automatic execution on EDIT	197, 261, 343
layer definitions.....	225	automatic execution on EXIT.....	200
translating into cell files	412	command for executing	115
Circle components		comments	120
angles	139	created by SHOW command	308
creating.....	132	created by TEMPLATE command	344
N_SIDES parameter.....	132	creating from journal files	100, 434
setting default N_SIDES	352	debugging.....	239
setting N_SIDES	128	directories searched to find.....	115
vertex calculation	127	editing cells in	194
Classroom Tutorials Manual.....	8	errors	118
Closing ICED		executing from menu	117
overview of related commands.....	113	executing in every cell.....	378, 380
Closing ICED	<i>See also</i> Terminating ICED	executing SHOW files.....	312
COLOR command	112, 167	executing with single keystroke.....	214
example in startup command file.....	30	general purpose	15
Color definitions		list of methods to execute automatically.....	91
in startup command file.....	29	location.....	17
stored in cell environment database	95	logging commands.....	234, 442
Colors		logging commands in.....	117
assigning to layers	220	M1NEWNUM.CMD.....	380
blinking	163	methods of executing.....	99

nesting	118	RAM parameter	80
overview of related commands	113	report on options	94
overview of startup cmd file	26	STARTUP parameter	82
passing data to programs	308, 316, 317	storing in project batch file	40
pausing in	265	syntax	51
PLOTIT.CMD	451	TIME_OUT parameter	83
recovering from errors	118	TMP parameter	84
saving macros in	214	VIEW_ONLY parameter	85
scratch layers used in	124	WINDOWS parameter	86
screen refresh during	289, 360	Command line parameters	
search path	48	overview	54
searching ICED_PATH for	49	Command syntax	
selecting components in	367	layer lists	106
simple explanation	14	notation used in manual	102
speeding execution	235, 360	verifying in journal file	268
technology-independent vs. technology-specific - 15		Commands	14
using batch file to execute	91	methods of executing	98
using care executing	311	overview of ICED™ commands	110
using GROUP in	207	repeating from the history	153
using OFFSET in ADD commands	127	Comments	120
using to execute utility	375	in batch files	41
Command line		in command files	34, 312
@file_name parameter	56	in log files	235
ALWAYS parameter	58	in options file	56
BATCH parameter	60	in SHOW command files	315
BEEP parameter	62	Component types	
CANCEL parameter	63	listed	122
D16 parameter	64	Components	
DEPTH obsolete parameter	70	aligning	253
exceeding 128 char limit	56	blanking	158
EXIT parameter	65	changing layer	338
ICED parameter	67	copying	175
input redirection	56	creating on layers	124
LEAVE parameter	69	cutting	180
LIMIT parameter	70	defined	94
LOG parameter	72	defining position of	125
MAXIMIZE parameter	73	deleting	181
MENU parameter	74	determining ids	307
NDIV parameter	76, 125	determining location	307
PAUSE parameter	77	determining type	307
QDIV parameter	78	editing	250, 317
QUIT parameter	79	labeling	145

-
- listing on certain layers.....309
 - merging243
 - mirroring176, 248
 - moving250
 - overview of modification commands111
 - overview of related commands.....110
 - partial vs. full selection299
 - passing data to other programs.....308
 - protecting from changes158, 284
 - reporting area, length313
 - reporting information on307
 - rotating177, 292
 - saving named list.....231
 - selecting295, 298
 - self-intersecting244
 - stretching250
 - unique ids296
 - Console window371
 - AUTOEXEC.BAT executed in373
 - closing374
 - commands to open.....114
 - opening.....375
 - opening editor without91
 - to display messages94
 - typing ICED.EXE command line51
 - using to execute batch file90
 - Console window messages
 - controlling pause in batch file77
 - Console window prompt.....184, 329
 - Console windows
 - opening in always cmd file.....35
 - Coordinates.....*See* Positions
 - reporting294
 - scaling all coordinates with UnStream426
 - COPY command.....111
 - COPY MS-DOS command.....42, 400
 - Copy protection438
 - Copy-edit cell libraries18
 - Copy-edit cell library.....45, 46
 - editing cell in.....21
 - Copying components175
 - Could not load cell message382
 - Current directory
 - CD command to change374, 376
 - Cursor
 - autopanning156
 - changing to spacing tool.....322
 - defining179, 183
 - using to digitize positions.....125, 318
 - using to select components300
 - CURSOR command.....112
 - in startup command file34
 - CUT command.....111
 - Cutting components180
 - D16 command line parameter64
 - Data
 - passing to other programs.....308
 - Data base units
 - defined.....76, 125
 - from ICED-1664
 - in CIF files.....416
 - in Stream files.....335
 - Data types
 - assigning to layers226
 - in Stream files.....333, 422
 - DEFAULT keyword
 - in LIST command.....231
 - Default layer
 - defined.....216
 - effect on spacing cursor.....220, 324
 - setting.....216, 351
 - Defaults
 - displaying on screen353
 - of layer parameters210
 - setting.....346, 351
 - setting default layer216
 - setting default layer216, 351
 - wire width.....219
 - DEFAULTS command351
 - Del2Cel utility97, 378, 382
 - Delay during redraw289
 - DELETE command111
 - Deleting

- removing one polygon from another245
- Deleting components181
- Depth
 - defined.....256
 - limiting display of nested cells by182, 364
- DEPTH obsolete command line parameter.....70
- Design rules checking188
 - spacing cursor220, 322
- Design Rules Checking.....*See* DRC
- Desktop shortcut.....451
 - creating new454
- Direct-edit cell libraries18, 46
- Direct-edit cell libraries45
- Directory
 - CD command to change current374, 376
- Displacements
 - measuring294
- Display..... *See* View window. *See* Screen display
- DISPLAY command.....112, 182
 - in always cmd file.....35
 - in startup command file.....34
- Distance
 - measuring294
- Dithering.....273
- DOC directory17
- Documentation
 - bitmap export281
- DOS command114, 184
 - using to execute utility375
- DOS console window
 - opening.....375
- DOS COPY command.....400
- DOS platform
 - does not support ICED.EXE13
- DOS shell184, 329
 - refreshing screen after command.....289
- DOSKEY system utility.....374
- DRC.....372
 - automating export.....456
 - finding self-intersecting shapes136
 - location of exported files.....19
 - selecting components by rule number296
- DRC (Design Rules Checker) utility
 - files used.....437
- DRC command113, 188
- DRC export
 - automating.....92
 - without opening editor.....79
- DRC.EXE12
- DRC_PATH372
- DRC_PATH environment variable190
- DRCEXPORT.BAT457
- DRCEXPORT.CMD79
- Drive letter
 - changing current in batch file457
- Dummy cell
 - using to edit cell in library.....20
- ED.CMD.....317
- EDIT CELL command
 - using to open cell in lib20
- EDIT command112, 192
- Editing cells192
 - comparison of commands.....193
 - in place258
 - nested cells258, 342
 - preventing changes85
 - with transformed coordinate system340
- Editor
 - opening Notepad from cmd line330
- ELSE command.....*See* Programmers Reference
- Embedded SELECT commands111
- ENTER.SUBCELL macro.....197, 260, 343
- Environment data base
 - defined.....95
 - not loaded for child cell.....192
 - not loaded for nested cell148
 - reporting344
- Environment variables
 - control of in console window331
 - creating in batch file376
 - creation in Windows 95,98.....376
 - defining with project batch file.....40
 - displying current values43

-
- during DOS command 185
 - referring to in batch file 49
 - summary 43
 - when required for utility 372
 - Error layers
 - selecting components on 297
 - Errors
 - bad wire outlines 136
 - corruption of swap file 84
 - Could not load cell xxx 382
 - during command files 118, 132
 - during CUT command 180
 - Error reserving virt mem for data pages 86, 87
 - from merged polygons 245
 - in CIF export 165, 227
 - in Stream export 334
 - in STREAM export 227
 - missing ICEDnAUX.EXE 438
 - missing pattern file 270
 - missing PDF file 270
 - Move failed for *n* component(s) 254
 - Radius too small -- change N_SIDES ? 128
 - recovering from using journal file 445
 - selecting failed components 304
 - self-intersecting polygons 136
 - There are *n* missing cells 97, 386
 - undefined layers on cif or stream export 227
 - use of XSELECT in command files 367
 - using SELECT FAIL 178
 - warning beep 62
 - Esc key
 - interrupting update of view window 289
 - using to cancel commands 63
 - Etching
 - removing one polygon from another 245
 - Exclamation mark
 - allowing in Stream structure name 409
 - Executable files
 - location 17
 - search path 49
 - Executing ICED by clicking cell 90
 - Executing ICED™ Utilities 371
 - EXIT command 100, 113, 198
 - used by AllCells 379
 - EXIT command line parameter 65
 - alternative methods 91
 - warning about other options 58
 - EXIT MS-DOS command 374
 - EXIT.ROOT macro 200, 230
 - EXIT.SUBCELL macro 200, 230
 - Exiting ICED
 - overview of related commands 113
 - Exiting ICED *See also* Terminating ICED
 - Export
 - automating 91, 450
 - automating CIF or Stream 458
 - automating DRC export 456
 - automating NLE export 456
 - bitmaps 281
 - layers in startup cmd file 32
 - overview of related commands 113
 - Extension
 - assigning file ext to ICED 90
 - External programs
 - executing when opening editor 35
 - Feature size
 - scaling all coordinates with UnStream 426
 - Files
 - redirecting utility output to file 376, 386
 - FILL command 112, 201
 - in startup command file 33
 - Fill Patterns 263, 437
 - assigning to layers 221
 - creating 402
 - effect on text display 143
 - enabling display of 201
 - fill mode affects line/text color 141
 - in plots 270
 - overview of related commands 112
 - setting in startup command file 31
 - Flattening cells 206
 - Font
 - used in Stream files 336

Index

- Fully selected components.....295, 299
- Function key assignments
 - in startup command file.....32
- Function keys.....213
 - assigning CANCEL to.....63
 - assigning every session.....58
- General purpose command files15
- Geometric data base148
 - defined.....95
- GLOBAL keyword
 - in LIST command.....231
- GLOBAL macro definition statement.....32
- Graphics
 - bitmap export281
- GRID command.....112, 203
 - in startup command file.....30
- Grid definitions
 - in startup command file.....30
- Grids
 - data stored in cell file148
 - display203
 - effect of resolution grid on cuts.....180
 - listing current settings309, 344
 - overview of related commands.....111
 - resolution, defined.....290
 - setting in startup command file30
 - snap, defined318
- Group command
 - using to copy components208
- GROUP command.....111, 206
- Guide *See* Spacing cursor
- Help
 - syntax for utilites.....377
- Hiding components
 - overview of related commands.....111
- Hierarchy
 - finding top-level file.....388
 - grouping/ungrouping cells.....206
- Highest-level cells
 - finding with ICTop.....388
- History153
 - of console commands374
- Home directories
 - adding to PATH50
 - defining in batch file.....44
- ICED command line option67, 88
- ICED desktop icon.....373, 374
- ICED.EXE12
 - launching from batch file.....40
 - location.....17
 - non-interactive mode91
 - requires environment14
 - specifying path on command line51
- ICED.EXE command line
 - environment variables must be defined first 43
 - storing in project batch file.....40
 - syntax51
- ICED_CMD_PATH115
 - defining in batch file.....48
- ICED_HOME373
 - adding to executable search path.....50
 - defining in batch file.....44
- ICED_PATH18, 379
 - defining in batch file.....45
 - defining in console window.....372
 - searching for command files.....49
 - used by ICTop389
 - used in ADD CELL146
- ICED_USER.....372, 373
- ICED-16
 - size of units.....64
- ICED5AUX.EXE438
- ICEDnAUX.EXE
 - avoiding timeout.....83
- ICList.....372, 384
- ICList.....373
- ICTop.....372, 376, 388
- ICTop.....373
- ICTree.....372, 377, 384
- ICTree.....373
- ICWIN.BAT12

example of using	23	Key assignments	
listing.....	40	in always cmd file.....	35
Id numbers		in startup command file	32
selecting from list	306	keeping up-to-date.....	58
Ids		listing current settings.....	344
determining component ids	307	Key assignments	213
listing with SHOW command.....	313	KEY command	114, 213
using to select components.....	296	Keywords.....	102
IF command.....	<i>See</i> Programmers Reference		
Import		Labeling components.....	145
CIF files.....	412	Lambda	416
finding top-level cell	389	Landscape orientation	
speeding display of large design.....	70	defined.....	275
Stream files.....	418	Large chips	
INITIALIZE command.....	110, 210	alternate memory method	67
in startup command file.....	32	speeding first time display.....	70
Intersecting		Launching ICED™	
removing one polygon from another	245	command line syntax	51
ITEM command.....	<i>See</i> Programmers Reference	Launching layout editor from batch file.....	40
		Layer *	
Joining components	<i>See</i> MERGE command	defined.....	210
JOURNAL command	100, 113, 211	LAYER * command	
Journal files	14, 20	in startup cmd files	32
automatic data recovery	442	Layer 0	
comments	120	assigning pen # for plot	222, 271
controlling buffer	72	defined.....	217
defined.....	99, 441	unblanking.....	159
editing to extract successful PLOT cmd.....	452	using in a layer_list.....	303
journaling during command files.....	442	using to select cells/arrays.....	285, 297
listing name	314	LAYER command	110, 216
location.....	19	overview of related commands.....	110
logging during command files	234	overview of related commands.....	112
recovering from mistakes	444	Layer correspondence file	
saved on exit.....	198, 211, 229	UnStream.....	419, 422
using to determine command syntax	268	Layer definitions	
using to recover from command file errors.....	118, 442	in startup command file	31
using to view error msg.....	65, 69, 79	stored in cell environment database.....	95
using to view error msgs.....	380	Layer lines	421
viewing.....	330	Layer lists	106
Justification codes	144	blanking.....	158
Justification codes for text.....	346	clearing names from	210
		examples.....	161

Index

- using in SELECT command.....303
- using to select components.....297
- Layer numbers
 - changing.....37
 - changing with command file.....380
- Layers
 - assigning cif parameters.....225
 - assigning colors.....167, 220
 - assigning fill patterns to.....201, 221
 - assigning plot parameters.....222
 - assigning stream parameters.....226
 - blanking.....160
 - changing component layer.....338
 - changing definitions in startup command file.....36
 - changing numbers.....37
 - changing numbers w/ cmd file.....380
 - changing spacing distance automatically.....324
 - CIF export.....165
 - default text size associated with layer.....142
 - definition in child cell ignored.....95
 - determining layer of components.....307
 - determining with Stream data type.....422
 - initializing parameters.....210
 - listing current settings.....344
 - listing names.....309
 - listing with SHOW command.....314
 - names.....210, 219
 - output in Stream files.....334
 - overview of related commands.....110
 - protecting from changes.....158, 285
 - reporting all components on.....309
 - selecting for plot.....274
 - setting default layer.....124, 351
 - setting default minimum space.....220
 - setting parameters.....217
 - swapping.....110
 - turning display of off/on.....366
 - used for scratch work.....124
 - using layer lists.....106
 - using to select components.....297
- Layout editor.....*See also* ICED.EXE
 - launching from batch file.....40
 - simple example of opening.....23
- Layout Editor Window.....94
- Layout Vs. Schematic.....*See* LVS
- LEAVE command.....100, 113, 229
 - used by AllCells.....379
- LEAVE command line parameter.....69
 - alternative methods.....91
 - warning about other options.....58
- Length
 - reporting component length.....313
- Library class.....*See* cell library class
- Library classes
 - defining in batch file.....45
- LIMIT command line parameter.....70
- Line components
 - color.....171
 - creating.....140
 - cutting.....180
 - fill mode affects color.....141
 - merging.....245
 - selecting.....298, 303
 - snap angle.....320
- LIST command.....113, 231
 - example with SEL LIST NEXT.....306
 - reporting named lists.....316
- LIST names.....232
- Lists
 - selecting components on.....306
- Lists of components
 - named lists.....231
 - reporting named lists.....316
 - select stack.....304
- LOCAL keyword
 - in LIST command.....231
- LOCAL_COPY keyword.....194
- LOG command.....113, 117, 234
- LOG command line parameter.....72
- Logging commands.....*See* Journal files
- Looping thru all subcells.....38
- Lower case
 - in Stream files.....407
 - in text components.....348

-
- LVS
 - automating export.....456
 - LVS.EXE.....12
 - Macro programming language.....8
 - Macro substitution
 - in comments120
 - Macros
 - defining in always cmd file35
 - ENTER.SUBCELL197, 260, 343
 - example of simple361
 - EXIT.ROOT.....200, 230
 - EXIT.SUBCELL.....200, 230
 - key assignment213
 - list definition231
 - list names.....232, 306
 - listing current settings308, 309, 314
 - saving214
 - system.....314, 315
 - user defined315
 - Map file for SFMap utility.....407
 - MARK_SUBCELLS*See* Programmers Reference
 - MAX_LAYER keyword.....337
 - Maximize command line option73
 - Maximum characters in dir name.....45, 48
 - MAXPATH45, 48
 - Measuring displacements.....294
 - Memory
 - optimizing262
 - reporting on usage262
 - Memory management
 - for large chips.....67
 - overview.....87
 - refining for physical memory80
 - windows virtual memory86
 - Menu
 - dimensions.....357
 - MENU command.....114
 - MENU command line parameter74
 - Menus
 - @START165
 - creating custom391
 - file names437
 - FILES, PLOT267
 - generating command syntax434
 - loading.....74, 241, 392
 - location.....17
 - nested view356
 - overriding menu in cell file74
 - reference stored in cell file95
 - temporarily loading submenus.....242
 - using to generate command syntax100
 - MERGE command.....111, 243
 - Mils
 - translating non-standard Stream files425
 - Minimum spacing requirements
 - setting in startup cmd file31
 - spacing cursor.....220, 322
 - MIRROR command.....111, 248
 - Mirrored text.....349
 - Mirroring
 - examples of, combined with rotation.....144
 - Mirroring components176, 248
 - Missing cell error messages97
 - MkMenu372, 391
 - MkPDF372, 393
 - MkPlot396
 - calling in batch file454
 - cancelling455
 - executing from the editor.....375
 - executing from the editor.....329
 - MkSti372, 402
 - Mouse*See* Cursor
 - Mouse buttons
 - using to cancel command63
 - MOVE command.....111, 250
 - selecting failed components.....304
 - Moving components or sides250
 - MS-DOS CD command374, 376
 - MS-DOS command interpreter184, 329, 371
 - MS-DOS console window
 - opening.....375
 - MS-DOS prompt.....371
 - MULTI_LINE_TEXT mode348

Index

- Multipage plots.....274
- Names
 - setting layer properties31, 219
- Names of cells96
- NDIV127, 335
 - can affect resolution grid.....291
 - defined.....76
 - displaying value.....309
 - importance in defining coordinates125
 - reporting value stored in cell384
- NDIV command line option76
 - avoiding warning message.....78
- Near box
 - setting size of.....255
 - using to select components.....302
- NEAR command.....111
 - in startup command file.....34
- Nested cells
 - blanking.....159
 - defined.....95
 - displaying labels.....182
 - displaying tree information.....384
 - editing196, 258, 342
 - grouping/ungrouping.....206
 - limiting display depth.....182, 364
 - listing components in.....309
 - report on opened94
 - stream file generation.....334
- Nested editing20
- Nested view menu.....126, 356
- Net List Extractor *See* NLE
- Network
 - printing.....399
 - using caution with TMP directory.....84
 - using command line options file.....57
- NEW command
 - used to create macros315
- NEW.CMD.....29
 - listing.....26
- NLE372
 - automating export.....456
- NLE.EXE..... 12
- Non-standard characters
 - allowing in Stream structure name 409
- Notation used in manual 102
- Notepad editor 330
- Offset
 - snap grid..... 320
 - using in ADD command 127
- Opening the layout editor
 - simple example..... 23
- Optional keywords..... 103
- Options 56
- OUTLINE command 112, 256
- Outlines
 - assigning pen # for plot 222
 - calculated from wire centerline 136
 - displaying cell/array outlines..... 183, 256
 - plotting 271, 276
 - reporting wire outlines..... 312
- Outlines..... *See also* Layer 0
- Output files
 - CIF 165
 - DRC 188
 - Stream 333
- Output redirection..... 386
- P_EDIT command 113, 258
 - using fill patterns 201
- PACK command..... 262
- Page size
 - defining plotters..... 394
- Panning view window
 - with arrow keys 153
 - with cursor..... 156
- Parameters
 - user-definable..... 344
- Parent cell
 - defined..... 95
 - returning to 100
- Partially selected components..... 295, 299, 304
- PATH..... 373

-
- PATH defining in console window.....372
 PATH environment variable.....49, 52, 185
 PATHEXT environment variable185
 PATTERN command110, 112, 263
 in startup command file.....33
 Pattern files.....201, 263, 402
 location.....17
 Patterns..... *See also* Fill patterns
 overview of related commands.....112
 setting file in startup cmd file.....33
 setting in startup command file31
 Patterns (fill).....263, 437. *See* Fill Patterns
 assigning to layers221
 creating.....402
 in plots.....270
 supplied with ICED™263
 PAUSE command.....113, 265
 example360
 PAUSE command line option.....77
 PDF - Plotter Definition File
 use in PLOT command.....270
 PDF - Plotter Definition Files
 creating or modifying393
 Pen numbers
 assigning to layers223
 used in PLOT command.....271
 Pgtxt directory17
 Physical memory87
 utilizing with alternate memory management80
 PIF file.....90
 Pitch
 setting snap grid319
 PLOT command113, 267
 automating export.....92
 extracting from journal file.....452
 using in batch mode.....451
 PLOTIT.CMD451
 Plots.....279
 assigning colors.....168
 assigning fill patterns.....223
 automating export.....92
 bitmap export281
 color mode.....272
 creating.....267
 default resolution and page size394
 Error-*n* layers assigned colors for which for which
 xxx_DOTS is undefined.....272
 Error-No data on output layers268
 file name.....278
 file names436
 labeling components.....145
 making text visible279
 multiple page274
 orientation276
 orienting text279
 paper size.....272
 PRESCALE, easiest way to plot area281
 printing/plotting.....396
 restricting array elements.....278
 restricting based on component size276
 restricting by cell depth277
 selecting area280
 setting layer properties in startup cmd file... 31
 units of scale.....279
 Plotter definition file.....*See* PDF
 Plotters
 defining393
 Plotting
 cell boundaries271
 Polygon components
 circles approximated with.....132
 closing131
 creating.....129
 cutting.....180
 merging.....245
 selecting.....298
 selecting sides.....295
 skipped positions.....130
 snap angle.....320
 spacing cursor.....322
 turned into boxes299
 used to approximate rings.....133
 used to approximate sectors140
 with nearly zero area132

- Portrait orientation
 - defined.....275
- Positions
 - calculated by ICED™127, 291, 318
 - defining in ADD command125
 - digitizing156
 - snap grid318
 - digitizing
 - changing view window while126, 356
 - closing polygons129, 131
 - in ADD command125
 - resolution grid290
 - snap angle320
 - how ICED™ stores76
 - range of possible values76
 - redundant removed.....130
 - reporting.....294
 - reporting component location.....307
 - rounding done by ADD command.....125, 127
 - storage of text component location.....145
 - typed by user125, 318
 - using OFFSET keyword in ADD command127
- PRESCALE
 - parameter of plot command.....281
- Printer definition files
 - location.....17
- Printers
 - defining393
- Printing
 - creating a plot file.....267
 - plots.....*See also* Plots
 - printing a plot file.....396
 - using a network399
- Program
 - command line syntax51
- Programs.....12
 - executing from editor command line329
 - executing when opening editor.....35
 - passing ICED™ data to.....308, 316
 - system search path.....49
- Project batch file.....14
 - assigning to file extension90
 - calling from another batch file..... 376
 - creating new 41
 - defined..... 40
 - defining env for utilities 372, 373, 374, 376
 - example of executing..... 23
 - executing before AllCells utility 379
 - executing automatically from icon 373
 - location..... 17
 - modifying for other purposes 450
 - sample listing..... 40
 - using as template 456
 - using in another batch file 453
 - using to open layout editor 40
- PROMPT command.*See* Programmers Reference
- PROTECT command..... 111, 284
- Protected cell library..... 19
- Protected libraries
 - editing cells with EDIT command..... 194
- Protected library classes 45
- Protection class..... *See* Cell library class
- Q:\ICWIN 105
- QDIV command line option..... 78
- Quote characters 213
- Quiet 62
- QUIT command..... 100, 113, 287
 - use in cmd file 453
 - used by AllCells 379
- QUIT command line parameter 79, 450
 - alternative methods..... 91
 - warning about other options 58
- Quitting ICED
 - overview of related commands..... 113
- Quitting ICED...287. *See also* Terminating ICED
- Quote characters 141
 - example of string containing quotes 141
- RAM command line option 67, 80, 88
- Read-only cell libraries..... 46
- Read-only library class*See* view-only library class
- Read-only mode.....*See* VIEW_ONLY
- Recovery.....*See* Journal files

-
- Rectangle *See* Box components
 - Redirecting utility output 376, 386
 - REDRAW command 114, 289
 - Refresh enabled parameter
 - defined 361
 - REM MS_DOS comment 41
 - REMOVE command 233. *See* Programmers Reference
 - Removing one polygon from another 245
 - Repeatable procedures 450
 - Repeating command lines
 - in console window 374
 - in layout editor 153
 - Reports
 - component area, length 313
 - environment variables 344
 - memory usage 262
 - nested cells 384
 - snap grid 321
 - using SHOW command 307
 - version used to create a cell 384
 - Required keywords 102
 - RESOLUTION command 112, 290
 - in startup command file 30
 - resets snap grid 318
 - Resolution grid
 - allowing calculated points to be off of 127
 - defined 290
 - defining cell origin on 207
 - displaying 291, 309
 - effect on calculated positions 127
 - effect on cuts 180
 - step stored in macro 314
 - stored in cell environment database 95
 - vs. snap grid 318
 - Resolution mode
 - effect on ADD command 127
 - setting 291
 - Resolution of plots 394
 - RETURN command *See* Programmers Reference
 - Ring components
 - setting default N_SIDES 352
 - vertex calculation 127
 - RING components
 - creating 133
 - Root cell 112
 - assigning on command line 51
 - defined 95, 146, 192
 - only cell environment used 97
 - using dummy as 20
 - Root layer
 - unblanking 159
 - ROTATE command 111, 292
 - Rotating components 177, 292
 - Rotation codes 177, 292
 - Rotations
 - examples of, combined with mirroring 144
 - Rounding error
 - in Stream import 426
 - scaling all coordinates with UnStream 426
 - RULER command 114, 294
 - Rules checking 188
 - Sample files 17
 - SAMPLE.DAT 403
 - SAMPLES directory 17
 - Saving
 - avoiding saving cell files 211
 - cell files 198, 229
 - macros 214
 - Scale
 - display 354, 357
 - limits detail on screen 363
 - scaling all coordinates with UnStream 426
 - Scratch files *See* swap files
 - Screen display
 - appearance of text 145
 - assigning fill patterns 221
 - autopanning 156
 - blanking components and layers 158
 - cell names 182
 - cell outlines 256
 - cursor 183
 - cursor coordinates 125
 - cursor types 179

- dimensions.....357
- during command files.....360
- fill mode affects line/text color.....141
- fill patterns263
- filled components201
- grids.....203
- limiting display by component size365
- nested view menu356
- of component/macro data316
- of template parameters344
- orientation of text147, 349
- overview of related commands.....112
- refresh289
- restoring last view window.356, 357, 358, 359
- scale.....357
- scale limits detail.....363
- show entire design.....355
- speed refresh by limiting detail363, 364
- speeding refresh151
- view window354
- Screen display:..... *See also* View window
- Search path for executable files49
- Search paths.....16
- Sector components
 - creating.....140
 - vertex calculation127
- Security key83
- SELECT command.....295
 - example in command file381
 - overview of related commands.....110
- Select marks.....296
- Selecting components
 - by bounding box.....300, 302
 - by bounding box.....303
 - by component type298
 - by DRC rule number296
 - by fail status304
 - by layer.....303
 - by layer.....297
 - by position.....303
 - by position.....300
 - by PUSH and POP304
 - by sides..... 295, 298, 299
 - by unique ids296
- catagories of tests296
- combining keywords.....302
- embedded SELECT commands.....111
- from named list..... 232, 306
- from ungrouped cell232
- in command files367
- in sample command file.....381
- in select box.....300
- new components150, 299
- using cursor303
- using cursor300
- using near box302
- using select stack304
- using shift select111
- Self-intersecting shapes136
- SET MS-DOS command42, 43
- Set of components
 - copying.....208
 - in named list231
 - saving on stack304
- Settings directory17
- SFDmp.....405
- SFMap407
- SFMap utility420, 421
- Shift key
 - using to select components111
- Shortcut.....*See also* Key assignments
 - creating new454
 - executing batch file from desktop.....90
- Shortcuts32
- SHOW command.....111, 307
 - sample output311
 - use caution executing output312
- Sides
 - moving.....251
 - selecting.....298
- Snap angle
 - defined.....320
 - effect on ADD LINE141
 - effect on ADD POLYGON129, 131

effect on ADD WIRE.....	136	auto-executing on open.....	58
no effect on ADD ARC.....	137	changing layer numbers.....	37
no effect on ADD CIRCLE.....	132	creating new cell without.....	52
SNAP command.....	112, 318	creating with TEMPLATE command.....	345
in startup command file.....	30	executing from menu.....	115
Snap grid.....	318	executing in existing cells.....	36
cursor snaps to.....	179	executing once editor is open.....	116
defining step.....	319	file name stored in macro.....	35
effect on ADD command.....	125	initializing layer parameters.....	210
offset.....	320	initializing resolution grid.....	290
scaling all coordinates with UnStream.....	426	LAYER commands in.....	217
step size related to snap angle.....	320	location.....	28
SONY portable computers.....	63	macros defined in.....	214
SPACER command.....	110, 322	MENU commands in.....	392
in startup command file.....	34	name stored in system macro.....	116
setting default space for layer.....	220	overview.....	26
Spacer mode cursor.....	322	specifying on the command line.....	82
Spacing cursor.....	322	used to assign stream layer numbers.....	334
changing automatically based on default layer.....	324	using before CIF export.....	165
setting default space for layer.....	220	using to assign cif/stream layer parameters.....	228
settings for layers in startup cmd file.....	31	STARTUP command line option.....	82
styles.....	326	STARTUP command line parameter	
toggleing on and off.....	327	alternative methods.....	91
Spawn command		using in addition to ALWAYS.....	58
using to execute utility.....	375	STARUP command line parameter	
SPAWN command.....	114, 329	combining with EXIT command.....	66
executing utility with.....	408	Stipple Patterns.....	437
Speed		creating.....	402
during command file execution.....	235	Storage	
improving performance.....	262	defining huge amount.....	67
reducing complexity of display.....	70	STREAM	
refreshing the screen.....	151	location of exported files.....	19
refreshing the screen.....	363	STREAM command.....	113, 333
Splitting components.....	180	example in command file.....	410
Stack SELECT commands.....	304	using to change layer numbers.....	38
START.CMD system macro.....	35	STREAM export	
Start->Run		automating.....	92
not good choice for executing utils.....	373	Stream file.....	435
Startup command file		assigning layer parameters.....	226
executing before Stream export.....	458	case of cell names.....	337
executing in every cell.....	381	clearing layer assignments.....	227
Startup command files		creating.....	333

- dumping405
- element flags428
- finding top-level cell389
- handling nested cells334
- importance of error file429
- importing.....418
- layer assignments not saved334
- layer names in startup cmd file.....32
- limitations.....429
- max layer number.....337
- missing cells.....97
- procedure for non-standard export409
- renaming structures407
- scale of units.....334
- speeding first time display.....70
- text height and font.....336
- units.....335
- use of boxes.....428
- Stream files
 - automating export.....458
- Stretching components.....250
- Structure lines.....420
- Structure names in Stream files408
- Subcell
 - editing protected library46
- Subcell editing.....20
 - comparison of commands.....193
 - EDIT command.....192
 - overview of related commands.....112
 - P_EDIT command.....258
 - T_EDIT command340
- Subcells13. *See* Nested cells
 - automatic cmds on exit.....200, 230
 - automatic cmds on open197
 - location of cell files.....19
- Submenus242
- Success comment in cmd files34
- SWAP CELLS command339
- SWAP command110, 111
 - executing in every cell.....381
- Swap file.....17, 84, 87
 - configuring for small86
 - defining alternate method67
- SWAP LAYERS command338
- Syntax
 - ICED.EXE command line51
 - layer lists106
 - used in manual.....102
- System macros315
 - listing settings.....308, 309, 314
 - START.CMD35
 - with startup command file name.....116
- T_EDIT command.....112, 340
- Tag numbers
 - using to select DRC shapes296
- TECH directory17, 22
- Technologies
 - migrating to new.....22
- Technology directories29
 - defining in batch file.....49
 - location.....17
 - overview22
- Technology directory.....458
- Technology files *See* Startup command files
- Technology-dependant command files15
 - location.....48
- Technology-independent support files
 - location.....17
- TEMPLATE command.....29, 344
 - using to save macros.....214
- Temporary directory68, 84, 89
- Temporary files.....84
- Terminating ICED100, 198, 229, 444
 - comparison of commands.....199
 - EXIT command198
 - JOURNAL command211
 - LEAVE command229
 - overview of related commands.....113
 - QUIT command.....287
- TEXT command110, 142, 346
 - in startup command file34
- Text components.....279
 - CIF export165

-
- color when fill is on.....143
 - colors.....171
 - creating.....141
 - default case.....348, 353
 - default justification.....353
 - default size219
 - display orientation.....144
 - fill mode affects color141
 - justification.....144
 - mirroring143
 - multiple lines.....141, 348
 - orientation348
 - placement affects ability to label.....145
 - plotting279
 - rotating143
 - selecting298
 - setting defaults346
 - size142
 - size in Stream files336
 - using lower case142
 - Text editor330
 - Text justification.....346
 - Text types
 - assigning to layers226
 - in Stream files333
 - TIME_OUT command line parameter.....83
 - TMP command line parameter68, 84, 89
 - TMP directory17, 84
 - Top-level cells
 - finding with ICTop.....388
 - TRACK_LAYERS option
 - of SPACER command.....220, 324
 - Tracking mode of SPACER command.....324
 - Triangle
 - using ADD POLYGON to create129
 - Tutor directory.....17
 - UNBLANK ALL command
 - example in command file381
 - UNBLANK command.....*See* BLANK command
 - UnCIF
 - run when editor is not open373
 - UnCIF utility
 - speeding first time edit70
 - UnCIF32412
 - UNDO command114, 350
 - after GROUP or UNGROUP209
 - UNGROUP
 - in LIST example.....232
 - UNGROUP command206
 - Ungrouping cells
 - blanking new components161
 - Units
 - in Stream files.....335
 - scale factor in plots.....279
 - used by ICED™76
 - used to store coordinates125
 - Unprintable characters
 - allowing in Stream structure name409
 - UNPROTECT ALL command
 - example in command file381
 - UNPROTECT command284
 - Unprotected cell library class*See* direct-edit
 - UNSELECT command295
 - UnStream418
 - non-standard structure names409
 - preserving case in structure names409
 - run when editor is not open373
 - using to change layer numbers38
 - UnStream utility
 - speeding first time edit70
 - USE command110, 351
 - in startup command file34
 - used to set default wire type135
 - User units
 - defined.....76, 125
 - in Stream files.....334
 - Stream file translation.....425
 - Utilities
 - aborting377
 - commands to open console.....114
 - defining environment for41
 - executing371
 - executing from console prompt373

- executing from editor374
- executing with batch file376
- executing with command file.....375
- ICList384
- ICTop388
- ICTree384
- location.....17
- MkMenu.....391
- MkPDF.....393
- MkPlot.....396
- MkSti.....402
- opening window for.....184, 329
- overview370
- redirecting output to file.....376, 386
- SFDmp405
- SFMap.....407
- syntax help377
- UnCIF32412
- UnStream418
- Version
 - reporting.....51, 308, 344, 384
- VIEW (ON | OFF) command.....113
- VIEW command.....112, 354
 - in startup command file.....33
- VIEW LIMIT command.....112, 363
 - in startup command file.....34
 - setting mode with command line option.....70
- VIEW ON/OFF command.....117, 360
- View only cell library45
- View window.....94, 354
 - autopanning.....156
 - changing while executing a command 126, 356
 - coordinates.....314
 - interrupting slow redraw289
 - limiting display by component size365
 - overview of related commands.....112
 - panning with arrow keys.....153
 - panning with cursor.....156
 - reducing complexity on open70
 - refresh289
 - saving358
 - scale.....357
 - scale used to enable view limit.....363
 - show entire design.....355
 - show selected components.....355
 - use last.....356, 357, 358, 359
 - zooming.....357
- VIEW_ONLY command line parameter85
- VIEW_ONLY keyword.....194
- View-only cell libraries.....18, 46
- View-only mode194
- Virtual memory.....67, 86, 87, 88
 - refining for physical memory80
 - swap file location.....84
- W95\$ENV.BAT file.....376
- Wait*See* PAUSE command
- Warning beep.....62
- WHILE command.....*See* Programmers Reference
 - example361
- White space characters104
- Wildcard characters315
- Window
 - background color.....29
 - changing size/mode73
 - closing console374
 - opening console window375
 - overview of related commands.....112
- WINDOWS command line option.....86, 87
- Windows File Explorer.....454
- Windows swap file.....67
- Windows virtual memory.....86
- Wire components.....136
 - creating.....134
 - cutting.....180
 - default width.....219
 - digitizing positions135
 - editing.....250
 - editing width.....317
 - effect of snap angle.....136
 - end types.....135
 - end types, in Stream files.....336
 - flush end types converted by UnStream428

labeling.....	145	changing in batch file	457
listing default width by layer	309	defining in batch file.....	47
listing outlines of.....	312	making current directory	23
merging	243	overview	19
outlines	136	setting in batch file	23
reporting length,area	313	setting with desktop shortcut	454
selecting	298, 303	types of directories to avoid	20
selecting sides.....	295	using temporary.....	20
self-intersecting	136	XCELLS.BAT	378, 380
setting default type	352	XSELECT command	113, 117, 367
setting layer properties	31	example in command file	381
setting pitch	319	XSELECT in journal files.....	368
snap angle.....	320	Zooming view window	
spacing cursor	322	changing scale affects view limit.....	365
used to approximate arcs	137	during command.....	126
width defined.....	134		
Working directory	48		
CD command to change	374, 376		

