PREPRESS MANUAL

PREPRESS version 2.1

Last revised by Lyle Ramshaw and Kerry A. LaPrade in September, 1980.

Available as: <AltoDocs>PrePress.Press (on your local IFS)

or as: [MAXC]<AltoDocs>PrePress.Press

or in raw form as: [MAXC]<AltoDocs>PrePressDocSources.Dm

PREPRESS MANUAL

Table of Contents

1 Introduction to the font world

PREPRESS is an Alto program for manipulating font files of various sorts (version 2.1 of PREPRESS will also run on a Dorado emulating an Alto). Before you can understand many of the weirder quirks of PREPRESS, it is crucial to have a good general understanding of how fonts are dealt with in PARC software. Section 1 will give you an overview of this fascinating subject, slanted towards what you need to know to understand PREPRESS. In Section 2, we will get back to our main topic: the questions of exactly what PREPRESS can do, and how to run it.

1.1 Fonts and font terminology

A *font* is a collection of character descriptions, indexed by character code. A font posesses various attributes, including its *family name*, its *point size*, and others. Each of these attributes has a collection of conventions surrounding it.

The *family name* of a font is a string that describes the particular design of character shapes used in the font. Two of the primary font families at PARC today are TIMESROMAN and HELVETICA. By convention, family names are written without spaces, and using only upper case letters.

Be careful not to confuse the family name of a font with the name of a file that happens to contain some representation of that font. The family name of a font is often used as part of a file name; for example, a file that contains a 10 point HELVETICA in .AL format (see Subsection 1.2 for formats) would conventionally be named "helvetica10.al". In this file name, as in all file names, upper case and lower case are interchangeable. Case only matters in the family name itself. Software may or may not believe that the family names "Helvetica" and "HELVETICA" are equivalent, and it is safest to punt the issue by sticking to upper case.

The *point size* of a font is some indication of how big the characters are. Most people seem to define the point size of a font as the distance in 72’nds of an inch between adjacent baselines of closely-spaced text. But there is a lot of room for argument about precisely what point size does or should mean. The size of fonts can also be measured in *micas* instead of points; a mica is 10 micrometers, or equivalently, 10^(5) meters. There are precisely 2540 micas in an inch. According to PARC software, there are exactly 72 points in an inch. The *mica size* of a font is simply its point size adjusted by the appropriate conversion factor.

The *face* describes a grab bag of other characteristics of a font; italic, bold, and condensed are all face attributes. There are two different encoding schemes in

common use for the face attributes. In file names, the only faces referred to are plain, bold, italic, and bold-italic; these are encoded as '', B'', I'', and BI'' respectively. In file names, the face code usually appears immediately after the point size. Thus, a 10 point HELVETICA in .AL format and with a bold face would be stored in the file named helvetica10b.al''.

Inside of PREPRESS, a fancier and more complete three letter code is used for face attributes. The first letter is either L'' for Light, M'' for Medium, or B'' for Bold; the second letter is either R'' for Roman or I'' for Italic; the third letter is either C'' for condensed, R'' for regular, or E'' for expanded. For example, a plain font has the three-letter face code MRR''; bold is BRR''; italic is MIR''; bold-expanded is BRE''.

As of version 1.13, the face code was extended somewhat. There is an optional fourth digit that describes the character code convention used in the font: the values are X'' for Xerox, A'' for ASCII, and O'' for other. In addition, numeric values of face are also allowed for TEX fonts; any integer or half-integer between 0 and 100 is a legal face, and specifies the logical size of the associated TEX font in points.

The *rotation* of a font refers to its orientation on the printed page. Rotation is measured in minutes of arc; there are 5400 minutes in 90 degrees. (The following discussion assumes that a standard sheet of paper is being held with its long axis vertically, as you are presumably holding the sheet upon which these words are printed.) A Dover printer scans the page from bottom to top, left to right. On raster devices with this scan pattern, a rotation of zero describes characters that (like the ones you are reading now) go horizontally from left to right across the page. Rotation increases in a counterclockwise direction; thus, on a Dover printer, characters rotated 5400 minutes would run vertically up the page. Most raster fonts that are encountered in practise are rotated by some multiple of 90 degrees, and hence some multiple of 5400 minutes. Arbitrary rotations are legal, from PREPRESS's point of view at least.

Devices that scan the physical page differently from a Dover have different rotation conventions. The most common example is a raster device that scans the page from left to right, top to bottom (the way that English is read). On such a device, characters that go from left to right across the page have a rotation of 5400 minutes, while characters that run vertically up the page are rotated 10800 minutes.

The *origin* of a character is a reference position that is used to describe the character's location on a page or display. Conventionally, the origin is located on the baseline at the left edge of the character. When a coordinate system is put on a raster, the PREPRESS convention is to locate the dots of the raster inside the squares defined by the lines of the coordinate system. Thus, the origin point is located at the

corner where four different raster dots touch, rather than in the middle of a raster dot.

The *width* of a character is a two-dimensional vector that describes the incremental translation that should take place on the page to determine the origin of the next character to be displayed in a string of text. The components of the width vector are floating point numbers in general. The width vector has two components to allow for fonts that are rotated by other than multiples of 90 degrees; if a font's rotation is a multiple of 90 degrees, one of the components of the width vector will be uniformly zero. Because we associate one width vector with each character, we cannot adjust the space after a character depending upon what character comes next. To say the same thing another way, we do not do spacing based on character pairs (kerning).

1.2 Font representations and formats

The characters in fonts are represented in one of two ways: (i) as a collection of cubic splines that specify the outline of the black area of the character, or (ii) as a raster (loosely referred to as a "bit map") that explicitly specifies the locations on a Cartesian grid that should be blackened to generate the character. A spline description of a character is resolution and device independent; the splines describe in a continuous, clean, mathematical universe the desired outlines of the character bodies. Rasters are a lower level and less general description of a character shape. First, a raster is tied to a particular combination of output device resolution and desired character size. Secondly, rasters are often adjusted somewhat to allow for the vagaries of a particular output device. If a printer tends to output less black than the raster requests, for example, the black areas of the raster can be expanded a trifle to compensate.

Perhaps the prime task of PREPRESS is to accept spline definitions of characters, and convert them into the rasters appropriate for a particular size, resolution, and output device. This process of going from splines to rasters is called *scan-conversion.*

In addition to scan-conversion, the PREPRESS program can perform many generally useful functions on fonts in various formats. Each of these formats has associated with it a filename extension; for example, a file containing a font in Carnegie Mellon'' format is usually given a name ending in the string .CU''. In fact, it is often convenient to use the extension as a name for the format: one speaks of a file in .CU format, or of a .CU file. Learn now the lore of formats and extensions:

.SF -- Contains spline definitions of characters in a LISP'ish text format, produced by a spline-editing program such as FRED.

.SD -- Contains spline definitions of characters in a more compact, binary form.

.AC -- Contains rasters for a font of a particular size at a particular resolution.

.OC -- Contains rasters for a font of a particular size at a particular resolution, but in a format distinct from .AC: see section 1.4.

.WD -- Contains width tables for a font.

.MD -- Contains metric information for a font: the widths and character dimensions are both described. This metric information is currently used by the document compiler TEX, and may be used by other formatters in the future.

.AL -- Contains an Alto font in a format suitable for use with the Convert operation.

.STRIKE -- Contains an Alto font in a format suitable for use with BitBlt. This original Strike format is unable to deal with characters that overhang either to the left of the origin, or to the right of the end of their width vector, however.

.KS -- Contains an Alto font in a format suitable for use with BitBlt. This new KernedStrike format cures the problem mentioned above.

.CU -- Contains a font in "Carnegie Mellon" format.

1.3 PrePress file format

Several of these file formats, in particular, the .SD, .AC, .OC, .WD, and .MD formats, are variants of a generic file format created and modified by PREPRESS, which, for want of a better name, we will call *PrePress format*. A file in PrePress format (called a *PrePress file*, for short) consists of an index followed by one or more segments; each segment gives some kind of information about a font, while the index includes pointers to all of the segments. In the simplest case, a PrePress file can contain an index with a single entry, followed by a single segment. In fact, a file in .AC format is more accurately described as a PrePress file with precisely one segment of type *Chars*. Similarly, an .SD file contains a single segment of type *Splines*, a .WD file contains a single segment of type *Widths*, and a .MD file contains a single segment of type *TexMetrics*. (There are two other types of segments, *OrbitChars* and *MultiChars*---but we will postpone discussion of them for a moment.) If a PrePress file contains a single segment of type *Chars*, we will sometimes refer to it as a *Chars* file, and similarly for each of the other four types of segments.

In general, a PrePress file can include many segments, of the same or different types; files with more than one segment are called *dictionaries*. A printing server running the PRESS program, for example, stores all of the fonts it knows about in a big

dictionary called Press.Fonts. This dictionary is a large file in PrePress format that contains *Splines* segments and/or *Chars* segments (and/or *OrbitChars* segments) for many fonts. The file Fonts.Widths is a dictionary of *Widths* segments, used by formatting programs such as BRAVO.

There are currently six different types of segments that can appear in a PrePress file. Four of them have already been mentioned:

Splines -- Compacted spline outlines for a font.

Widths -- A table of width data about a font.

TexMetrics -- A differently formatted table of width and size data for a font, currently used by TEX (a document compiler).

Chars -- Rasters for a font.

The other two, *OrbitChars* and *MultiChars*, demand rather more complex explanations.

1.4 The type *OrbitChars*

A raster for a character is simply a two-dimensional array of bits. The natural way to store such an array is in some variety of row-major order. The rows of a raster are known as *scanlines*, since the successive bits of a row are used to control one of the scans of the output device. In a *Chars* segment, each scanline is stored 16 bits per word in successive words, with the first bit of each new scanline constrained to start a new word. Thus, if a raster has scanlines that are 17 bits long, each of them will occupy two full words, with the last 15 bits of the second word wasted.

Some printers driven by Altos have a special piece of supporting hardware called an Orbit; for example, a Dover printer is usually driven by an Alto equipped with an Orbit. This piece of hardware is designed to help the program deal rapidly with rasters. In particular, the Orbit is able to merge lots of little rasters from various places in memory to produce one large raster for output to a printing engine. An Orbit expects its input rasters to be stored scanline by scanline in successive words of memory *not aligned on word boundaries!* That is, each scanline should start immediately after the previous scanline ends, regardless of where in a word this occurs; no bits are unused.

This provides the background for the *OrbitChars* type of segment. An *OrbitChars* segment and a *Chars* segment both consist of rasters for a font at a particular size and resolution. The primary difference is that the successive scanlines of the rasters are aligned on word boundaries in a *Chars* segment, and not aligned in an *OrbitChars* segment. The header information at the beginning of the raster is also in

a different format, but that is a minor point. PREPRESS prefers to work with rasters of type *Chars* whenever possible. In particular, most PREPRESS commands that use rasters as input (such as Show, Edit, Rotate, Grow, and Shrink) will only accept *Chars* rasters, not *OrbitChars* ones. In fact, the only consumers of the *OrbitChars* rasters are those printers that use an Orbit interface. Since the Altos that drive Dovers use Orbits, the dictionary of fonts used by the programs PRESS and SPRUCE when driving a Dover should have raster segments of type *OrbitChars*, rather than of type *Chars*.

Confusing as this dual format may seem, it is really not very painful, because PREPRESS includes commands that translate a font both ways between the two formats: the OrbitFormat command will convert *Chars* to *OrbitChars*, and the DeOrbitize command will convert *OrbitChars* to *Chars*.

If the name of a font file ends with the extension .OC, that file, by convention, should contain a single segment of type *OrbitChars*. Think of the following mnemonics: .AC means a *Chars* segment and could be read Aligned Chars'' while .OC implies an *OrbitChars* segment and could be read Orbitized Chars''. When the *OrbitChars* type was new, many files that would be called .OC files today were given the extension .AC instead. Some of these files may still be hanging around, or you may just meet font files that don't follow the standard naming conventions at all. If you have a PrePress file with a single segment, you can easily determine what type of segment it is (regardless of the file name) by the following technique: invoke PREPRESS, select the Show command, and type the name of the file into the Source File window. PREPRESS will display lots of interesting data about the file in various places; in particular, the type of the segment will appear in the File Type window (see Figure 2 1).

1.5 The type *MultiChars*

There is now only one more segment type to discuss, the type *MultiChars*. This type is a recent addition to PREPRESS, and it is not yet well integrated into the PREPRESS world. But the basic idea is as follows.

Over the years, fonts tend to be updated. For example, the basic PARC roman fonts have been revised/improved several times. With each new revision, the characters look a little more like the ones that real-world publishers and printers produce. In particular, in each new revision, the characters tend to be thinner and more packed together. Consider what will happen, now, when an old Press file is sent to a printer using the new fonts. A problem arises because the old Press file has buried inside it knowledge of the widths of characters: the formatter has to know these widths to decide where to break lines, for example. Since the width information that the formatter used is no longer correct, something bad is bound to happen.

In the Press file, the successive characters of a line are output in blocks whose left edge position is prescribed in an absolute coordinate system. Since the characters in that block are now thinner than the formatting program thought that they would be, the block runs out too soon, and all the extra space is concentrated at the right end of the block. Since these blocks often run up to half a line or so, this makes the right margin of justified text look unacceptably ragged. The new characters, mind you, are not all that much thinner than the old ones; no one would notice the difference if the extra space in the line could be spread out more evenly. But concentrating all the extra space at a few places, in particular, at the ends of lines, makes it painfully obvious.

The type *MultiChars* is part of a fix for this problem. Each system that writes Press files now timestamps them with the date of their creation. The printing servers are expected to remember the old versions of each font, as well as the current one; each Press file should be printed with the versions of its fonts which were current when that Press file was created. In fact, to save space, the printers don't remember the old rasters: the new rasters presumably look better anyway. But they do remember the old widths. When an old Press file comes in, the printer uses the new rasters, but positions them according to the old widths; the resulting documents look fine, since the extra space is spread out uniformly over the entire line. [All printing servers should check the timestamp of Press files and act appropriately; SPRUCE actually does so, but PRESS currently does not.]

A *MultiChars* segment is just like an *OrbitChars* segment, except that it allows for more than one table of widths in fact, the expression *MultiChars* is really short for MultiWidthChars''. There is space in a *MultiChars* segment for up to four different width tables with their associated dates. The PREPRESS command ReviseWidths allows you to replace the rasters in a *MultiChars* segment with a new and improved set of rasters, and to add the widths for these new rasters at the front of the current table of widths.

But ReviseWidths is one of the few commands that understands about *MultiChars* segments at all. To date, most PREPRESS commands cannot handle *MultiChars* segments, and in fact, don't even know of their existence. There are no capabilities for Rotating, Editing, Showing, or otherwise dealing with *MultiChars* segments. If you try to Extract a *MultiChars* segment from a dictionary, the extract operation throws away all but the most recent widths, and outputs the resulting thing as an *OrbitChars* segment. The current kludge status of *MultiChars* may change someday.

1.6 Pointers to further information on fonts

For a lower level introduction to what font files actually look like on a word by word basis, for a definition of such interesting terms as *bounding box*, and especially for a

great figure on the last page, you should read the current version of the document titled something like "Font Representations and Formats." It is stored on the file [Maxc1]<PrintingDocs>FontFormats.Press.

2 Things to know before using PREPRESS

The PREPRESS program is available as the file <Alto>Prepress.Run (You should be able to find it on your local IFS, but if not try [Maxc1].). There are two ways to give commands to PREPRESS: through interactions with a menu, or through directives typed on the command line. For most purposes, interacting with the menu is the style of choice. The command line mode of operation is a relic from the early development of PREPRESS. It has been retained because of its usefulness in a particular situation: you occasionally want to direct PREPRESS to perform a long sequence of similar operations, say when building a large new dictionary of rasters from scratch. Doing this through the menu interface is tedious, since you must wait for the previous operation to complete before issuing the request for the next operation. Using the command line interface, you can construct a long command line as the text file CommandFile.CM, invoke PREPRESS with the command Prepress @CommandFile.CM@'', and go out for a cup of coffee. Because the command line interface is a relic of history, it has its own funny conventions and assumptions. Most of the following text will describe the menu interface. The description of what you can get through command line requests, the appropriate syntax, and related issues are dealt with in an Appendix.

To use PREPRESS through the menu interface, one merely types "Prepress.Run" to the Executive. After a little thrashing around, a command menu will appear, which we shall call the *Main Menu*.

Note: You may wish to refer to Figure 2 1, which is an illustration of the PREPRESS Main Menu, as you read the remainder of this section. Those blocks in the menu requiring information to be entered, such as names of files, or the numbers needed to perform one conversion or another, have a horizontal (and in the case of some "Dictionary" operations, vertical) line dividing the block into two parts. One part contains the title of the information required and the other part is reserved for entering the needed information. In some cases, PREPRESS offers default information for these blocks but you may change this information if you desire.

The various commands on the menu are invoked by moving the mouse until the cursor is inside the desired block and then depressing any button on the mouse. The cursor is in the shape of a small arrow pointing to approximately the 10 o'clock position. Depending upon which command is invoked, certain of the blocks on the menu may turn black to prompt you regarding what additional information PREPRESS requires in order to perform the command selected.

It is mandatory that information be provided to some of the blocks and optional that information be provided to others. The instructions that follow in other sections of this document identify the mandatory and optional blocks. For example one of the optional blocks, "Incline," may ask for the percentage of *inclination* the characters in the file are to be tilted in order to create a font which looks (something) like italics. The default inclination is, of course, zero.

Most information is entered by selecting a box, typing in the data on the keyboard, and terminating with RETURN. For a few of the blocks, however, only one of two answers are possible. In these cases, the blocks will always contain one of the two possible choices. If the opposite choice is desired, it is invoked by moving the cursor into that block and depressing any button on the mouse. The other possible choice will then appear in that block. Each time any button on the mouse is depressed with the cursor in one of those blocks, the opposite choice will appear. In the case of the "Angle" block there are four possible choices. If one of the other three choices is desired, it is invoked by moving the cursor into the Angle block and depressing one of the three mouse buttons.

At the top of the main menu is a lightly outlined block that can contain a single line of text. This block is used by PREPRESS to prompt you concerning information that you have decided to enter. For example, if you indicate with the mouse that you would like to enter an inclination for synthetic italics, the prompt window will tell you what units to use in measuring the inclination. Once you have entered the desired inclination, the prompting message will disappear. Other instructions and information for the operator are displayed in the message area at the very top of the screen.

There are currently four blocks on the menu pertaining to file names. These are "Source file," "Output file," "Dictionary" (sometimes referred to as "Big") file, and "Background" file. Each command on the menu involves either 1 or 2 of these blocks. In general (but not always) invoking a command with either the *left* or *middle* mouse button will fill in the appropriate file blocks with the *default* file name for the appropriate file type, e. g., "SDTemp" for a (one-segmented) spline definitions file. Invoking with the *right* mouse button fills in the appropriate block or blocks with the *most recently used* file name (other than the default name) for the appropriate type. After invoking a command, the same mouse button functions apply if you desire to change a file block on an individual basis. The only exceptions to this rule (known to this author) are the Background File (used only by the Edit Command), commands involving STRIKE and CU files, and some commands which allow more than one legal file format, in particular, the Rename command and most of the dictionary commands. Note that the standard default name "ACTemp" is used for both the *Chars* and *OrbitChars* file formats. In many situations you may be able to save

keystrokes by clever mouse-clicking.

After all of the mandatory and optional data for each operation is entered into the blocks of the menu, the procedure being accomplished is started by moving the cursor into the "DO IT" block at the bottom of the menu and depressing any button on the mouse. When the procedure completes, the main menu will return, and you may select another operation; to return to the Alto Executive, invoke the "QUIT" block.

If you have selected an operation and, before invoking "DO IT", you change your mind, you can simply select another operation; the menu will be adjusted to prompt you for the information needed by this new operation. Alternatively, you could return to the Alto Executive without performing any operation by invoking "QUIT". By the way, it doesn’t work to set up the parameters first, and then select the operation: in most cases, selecting the operation flushes any old choices for relevant parameters and replaces then with the standard defaults, so you should first select the operation, then set the parameters.

The blocks in the uppermost chunk of the menu are used to describe the contents of the source file as well as to accept parameters from the user. In particular, if PREPRESS is currently in a state in which it expects the Source file to be a file in PrePress format with a single segment, and you enter a Source file name into the appropriate block, PREPRESS will fill in the family name, the face, the type of that single segment, the point size, the rotation, and (if appropriate) the resolutions by reading the source file that you have named. If the Source file has more than one segment, its type will be displayed as "Dictionary," and the family name of the first segment in the file will be displayed, but all other displayed characteristics are meaningless. You can put PREPRESS into a state where it expects a single segment Prepress format file as input by selecting the Show command; after examining the characteristics of the file in question, you may select another command or invoke "QUIT" as you see fit.

The rest of this Manual will be devoted to explicit descriptions of each of the available commands, at the rate of about one command per section. But before we begin that large topic, perhaps a word of information about PrePress.Scratch is in order. In addition to all of the official PREPRESS and non-PREPRESS files that are discussed in the descriptions of the commands, PREPRESS sometimes just needs some scratch file space to keep temporary results. When it feels this need, PREPRESS uses the file named PrePress.Scratch; if such a file already exists, PREPRESS overwrites it, otherwise PREPRESS creates such a file. This can cause unexpected results if you have a Trident disk drive on line. You may delete PrePress.Scratch after leaving PREPRESS if you like.

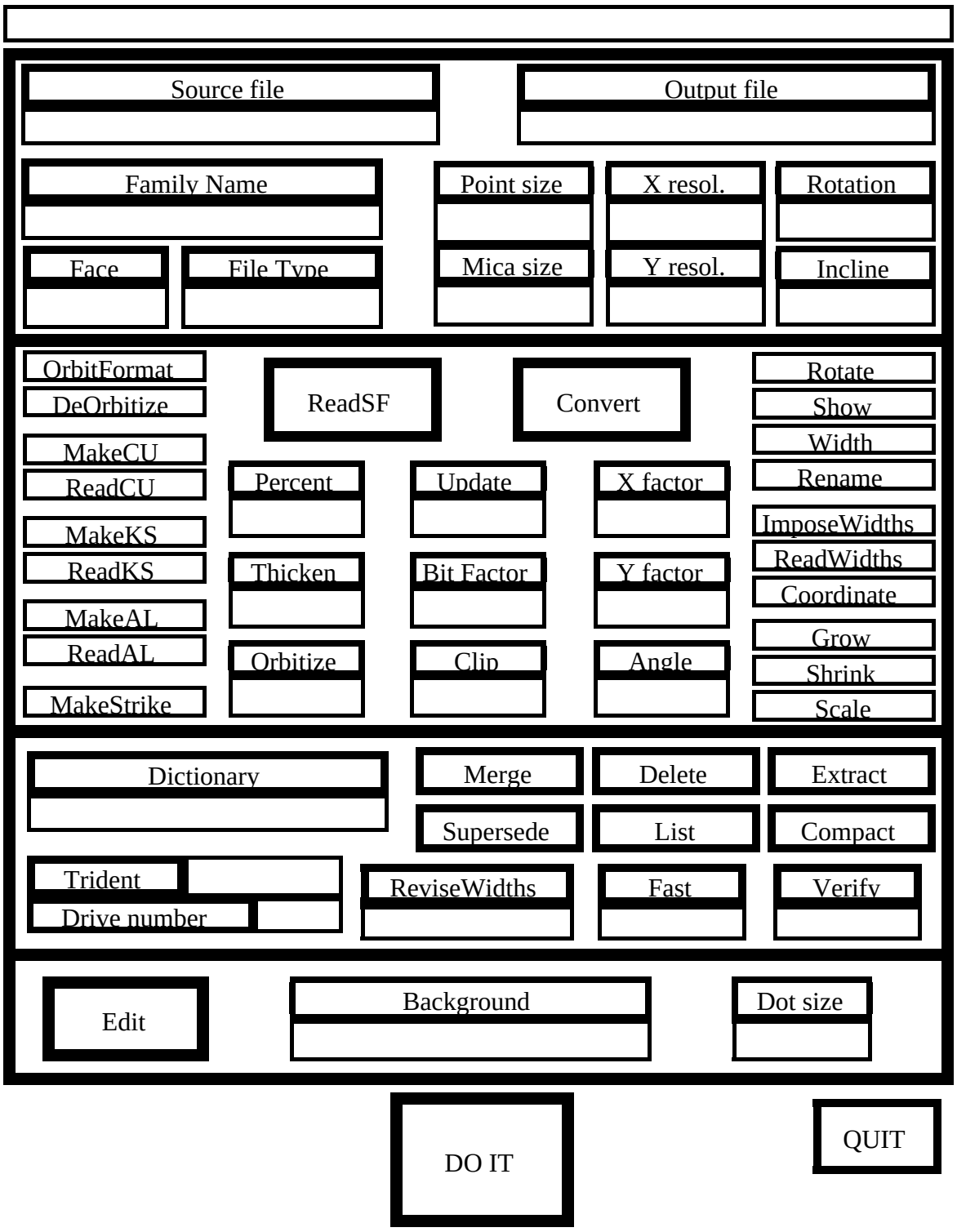

Figure 2-1: PREPRESS Main Menu

3 ReadSF: Reading .SF files produced by FRED

The command ReadSF takes the bulky .SF spline files produced by the spline editor FRED, and reformats them in the more compact .SD format, which can then be used as input to the scan-converting command. That is, ReadSF goes from .SF files to a *Splines* file.

PREPRESS accepts spline font (.SF) files created with FRED. In reading in and converting the file, problems can occur if the .SF file is not properly formed. One area of potential trouble is in entering width and fiducial information and the data required under the "Miscellaneous" block of FRED. Width and fiducial information which appear to PREPRESS to be out of bounds or in some way improperly formed may cause the .SF file not to be read in or if read in, not able to be converted into other types of files. PREPRESS tends to be unforgiving in these matters. In some cases, messages will appear on the screen alerting the operator of where the problem lies, however, in other cases, PREPRESS will go into Swat or just "hang", requiring the operator to either boot or Swat to recover.

In entering the data required by the "Miscellaneous" command of FRED, make sure that the family name is the same for all of the characters in the same .SF file. If the family names are not the same, a message to that effect will be displayed during the conversion process in PREPRESS.

Generally, if a problem consistently occurs during the first two major steps of PREPRESS, namely reading the .SF file and converting to an .AC file, the problems can often be traced to an error in the .SF file created in FRED.

The ReadSF procedure (in conjunction with the Update flag) allows a number of separate .SF files to be merged into a single compact .SD file. This single compact file is usually called SDTemp; however, the name of the file can be changed if desired. To read in an .SF file proceed as follows:

a. Type Prepress.Run to the System Executive, terminated by RETURN, and wait until the main command menu appears.

b. Move the cursor to the block labeled "ReadSF" and depress the *left* button on the mouse. The background in the ReadSF block will turn black along with the backgrounds in some additional blocks. The color change is used to prompt you regarding what information is required.

c. Move the cursor to the "Source file" block and depress any button on the mouse. Type in the name of the source file (which for clarity should have the extension .SF) and terminate with RETURN.

NOTE

Once you have entered this name, it is remembered until you either "QUIT" or type in another "Source file" name for a ReadSF operation (since ReadSF is the *only* PREPRESS operation involving .SF files). The name is automatically recalled whenever you invoke ReadSF" with the *right* mouse button. Similarly, it is also recalled when you invoke ReadSF with *any* mouse button and then select the Source file block with the *right* button.

d. PREPRESS offers the default file name SDTemp (assuming you used the *left* mouse button) for the output .SD file. If you want to change the name of the output file, move the cursor to the "Output file" block and depress any button on the mouse. Type in the name of the output file (ending it with .SD) and terminate with RETURN.

NOTE

Once you have entered this name, it is remembered until you either "QUIT" or type in another name in a "Source file" or "Output file" block while PREPRESS is in a state expecting a *Splines* file in that block. The name is automatically recalled when you invoke with the *right* mouse button a command which involves a *Splines* file as either Source or Output. Similarly, it is recalled when you invoke such a command with *any* mouse button and then select the relevant Source or Output file block with the *right* button.

e. The block labeled "Update" has "false" as its default value. If the Update flag is false, the original contents of the output .SD file are ignored: the output file after the ReadSF command will contain compacted representations of exactly those characters defined by the current .SF input file. In many situations, however, you want instead to add some new characters to an existing .SD file. For example, you might want to merge the characters defined in several .SF files into a single .SD file. Or you might want to replace the definitions of one or more characters in an .SD file with new definitions from an .SF file, without altering any other characters. In these situations, you should set the Update flag to "true". Move the cursor to the block labeled "Update" and depress any button on the mouse. The word "false" which normally is in the bottom half of the block will change to "true." Now, when the ReadSF is performed, the character definitions in the .SF file will be taken as additions or edits to the current .SD file. If the .SF file defines a character that has no current definition in the .SD file, the .SF definition will be added to the .SD file. If the .SF file redefines a character that is already defined in the .SD file,

the .SF definition will replace the current definition in the .SD file; in this case, a warning mesage will appear stating that characters have been multiply defined.

NOTE

Steps f and g are optional commands which may be skipped if desired. Step f describes how to slant the .SF file characters to form pseudo-italics; and step g describes how to expand or condense characters.

f. As the splines are read from the .SF files, it is possible at this time to change the fonts into pseudo-italics by slanting each character to the right. This is called "Incline" and is specified as a percentage. To slant all of the characters in the file, move the cursor to the block labeled "Incline" and depress any button on the mouse. The background on the bottom half of the block will turn black and a message will appear requesting the percentage of inclination. The incline is measured as a percentage of the *y* coordinate to be added to the *x* coordinate. Type in the percentage (typically 20% is more than sufficient) and terminate with RETURN. The number entered will appear in the specified block and the block background will return to white.

NOTE

The incline command is not a true representation of italics, but only resembles this type form by slanting each character to the right by the specified percentage.

g. As with the incline feature, characters may be condensed or expanded as they are read in from the .SF files. The "X factor" and "Y factor" blocks are used for this purpose. These functions are normally set with scale factors of 1.0 (no scaling). The X factor scales the font in the horizontal direction, and the Y factor scales the font in the vertical direction. To condense a font, move the cursor to the block labeled X factor and depress any button on the mouse. The background in the bottom half of the block will turn black and a message will request that the scale factor be typed in. Terminate with RETURN. If the font is to be condensed to 3/4 as wide as specified in the .SF file, type in .75. To expand the character, you might enter 1.25.

The Y factor block accomplishes the same condensing or expanding in the vertical direction, however it should be used with some care as values other than 1 in this block will alter the point size of a font. For regular characters of the alphabet, the Y factor is seldom used, however, for

special characters or logos, it may be desirable to expand or condense in the vertical direction.

h. When all of the information is entered, move the cursor to the block labeled "DO IT" and depress any button on the mouse to complete the command.

NOTE

Several kinds of errors can be generated during the "Read SF" operation. Warnings are generated if characters with different family names or different face designations are read into the same file from different .SF files. Messages will appear on the screen alerting the operator of these occurrences. Family name or different face designation anomalies can be corrected by returning to the FRED program and correcting these errors by invoking the Miscellaneous block.

After the Read SF operation is complete, the .SF files are no longer needed for PREPRESS operations and may be stored or archived.

4 Convert: from splines to rasters

The Convert operation allows you to scan-convert the spline encodings contained in the SDTemp file just created in section 3 (or other .SD file specified by the operator) into an .AC file; that is, Convert takes a *Splines* file and produces a *Chars* file. The default name ACTemp is provided for the output file, but you may change the name if you wish. Of course, since you are moving to a raster representation, you must specify the size, resolution, and rotation that you desire. There is abolutely no guarantee that scan-converting splines at a particular size and resolution will produce rasters that are acceptable, or even vaguely legible. If you are trying to produce good-looking small rasters for a font, you can improve the results of scan-conversion somewhat by a sampling process to be described later (see the Scale command). But you will probably end up having to edit the resulting rasters by hand it you really want them to look good (see the Edit command). At higher resolutions, a straightforward scan-conversion is pretty likely to produce acceptable rasters.

To convert a *Splines* file to a *Chars* file, proceed as follows:

a. Move the cursor to the block labeled "Convert" and depress any button on the mouse. The background in the Convert block will turn black along with the background in some of the other blocks.

b. If you invoked Convert with either the *left* or *middle* button, the Source file will automatically read "SDTemp." If you instead invoked Convert

with the *right* mouse button, the "Source file" will automatically read whatever you last typed (other than "SDTemp) into a Source or Output file block while PREPRESS was in a state expecting a *Splines* file in that block. If this is the wrong Source file name, move the cursor to the Source file block and depress *any* button on the mouse, type in the correct file name (For clarity, this should be either "SDTemp" or a name ending with ".SD".), and terminate with RETURN.

c. If you invoked Convert with either the *left* or *middle* button, the Output file will automatically read "ACTemp." If you instead invoked Convert with the *right* mouse button, the "Output file" will automatically read whatever you last typed (other than "ACTemp) into a Source, Output, or Background file block while PREPRESS was in a state expecting a *Chars* file in that block. If this is the wrong "Output file" name, move the cursor to the "Output file" block and depress *any* button on the mouse, type in the correct file name (For clarity, this should be either "ACTemp" or a name ending with ".AC".), and terminate with RETURN.

d. The desired point size of the font or character must be entered in the block labeled "Point size." This information is mandatory. Invoking this block with the cursor will produce a message which requests "Size in points". Type in the size (there are approximately 72 points to the inch) and terminate with RETURN. The number will appear in the Point size block and the size in micas will be computed by PREPRESS and appear in the block labeled Mica size. An alternate method of entering the font size is to use the block labeled "Mica size" and enter the vertical size in micas, if known. (A mica is 10 micrometers and there are 2540 micas to the inch.)

e. The resolution of the printing device that will use the character must be entered next. This information is mandatory. Since the horizontal (X) and vertical (Y) resolution of the same printer could be different, two blocks, one labeled "X resol." and another labeled "Y resol." are provided for that purpose. Enter the appropriate resolution for the desired device. Since the X and Y resolutions of most printers are the same, for convenience, the number entered in the "X resol" box is automatically entered in the "Y resol" box; however, you may change the Y value if you wish. For Dovers, the resolution in both the X and Y direction is 384 bits per inch. The resolution of the Alto screen is pegged by various authorities at anywhere between 72 and 80 bits per inch. One common value is 72 bpi.

NOTE

Steps f through i are optional commands which may be skipped if desired. Step f describes how to create pseudo-italics; step g describes how to rotate characters; step h describes the Thicken flag; and step i describes the Orbitize flag.

f. If it is desired to create pseudo-italics of all the characters in a font, it may be accomplished during this operation. (This was also an option during the previous step of reading the SF files.) To slant all of the characters in the file, move the cursor to the block labeled "Incline" and depress any button on the mouse. The background on the bottom half of the block will turn black and a message will appear requesting the percentage of incline. The incline is measured as a percentage of the *y* coordinate to be added to the *x* coordinate. Type in the percentage (typically 20% is more than enough) and terminate with RETURN. The number entered will appear in the block and the background will return to white.

NOTE

The incline command is not a true representation of italics, but only resembles this type form by slanting each character to the right by the specified percentage.

g. Characters may be rotated, if desired, by entering a number in the block labeled "Rotation." The number entered is the number of minutes of arc that each character is to be rotated about the origin in a counterclockwise direction. Since there are 60 minutes in each degree, to create characters that will run up a Dover page (that is, to rotate the font 90 degrees in a counterclockwise direction) enter the number 5400 (90 times 60) in the block labeled "Rotation." Rotations by other than multiples of 5400 minutes are unusual, but legal; of course, neither component of the resulting width vectors will be zero.

h. The Thicken flag is normally "false". When Convert runs in this mode, thin lines of black in the character sometimes have gaps, caused by the effects of discretization. Setting the Thicken flag to "true" changes the discretization algorithm in such a way that bits are more likely to be made black. This tends to make low resolution rasters more bold, but sometimes a little better looking. More precisely, the output of Convert with the Thicken flag set is the result of OR’ing together the raster that Convert would normally produce with that same raster raised up by one raster dot.

NOTE

The Grow command (described below) is a more drastic way to create a **bolder** raster.

i. The Convert operation usually produces a result of type *Chars*; setting the Orbitize flag will cause the output to be of type *OrbitChars* instead. One could achieve the identical effect by running Convert without setting the Orbitize flag and then invoking the OrbitFormat command (see below).

j. When all of the information is entered, move the cursor to the block labeled "DO IT" and depress any button on the mouse to invoke the command.

5 Show: examining an entire font

The Show command allows you to examine an entire font of rasters, one character at a time. Unfortunately, there is no way to change the order in which the characters are shown. In order to examine a font with more flexibility, you should use the Edit command described below. Both the Show and Edit commands only work on *Chars* fonts, that is, non-orbitized rasters.

NOTE

If you are running PrePress on a Dorado emulating an Alto, the display in the Show command will be smaller than it should, and will have garbage below it. This happens because the Dorado display microcode does not implement the low-resolution mode of the Alto display. Fortunately, a Dorado is fast enough that there is no reason to avoid the more powerful Edit command.

To Show a *Chars* file, proceed as follows:

a. Move the cursor to the block labeled "Show" and depress any button on the mouse. The titles of the "Show" and "Source file" blocks will turn to white on black.

b. If you invoked "Show" with either the *left* or *middle* button, the Source file will automatically read "ACTemp." If you instead invoked Convert with the *right* mouse button, the Source file will automatically read whatever you last typed (other than "ACTemp") into a Source, Output, or Background file block while PREPRESS was in a state expecting a *Chars* file in that block. If this is the wrong Source file name, move the cursor to the Source file block and depress *any* button on the mouse, type in the correct file name (For clarity, this should be either "ACTemp" or a name ending with ".AC".), and terminate with RETURN.

c. Move the cursor to the block labeled "DO IT" and depress any button on the mouse.

d. The menu will disappear and the first character in the AC file will appear on the screen along with its width information.

e. Each character is individually displayed by depressing the RETURN key. Depressing the RETURN key after the *last* character is shown or depressing the DEL key *at any time* returns PREPRESS to the main menu.

6 Edit: examining and altering rasters

The "Edit" command allows you to selectively display and edit the characters of a *Chars* file (whose name conventionally has the extension .AC). This editing process is called *face editing* and is accomplished by changing the status of bits in the raster back and forth between black and white, and by adjusting if necessary the origin point and width vector of the character. The characters of the font being edited can also be displayed in a string or in words in order to evaluate their appearance and spacing. Good face editing is art, not science; especially in low resolution fonts, there just aren't a whole lot of bits to play with, and producing good-looking characters takes talent and work. This is the stage where most of the aesthetic judgements regarding appearance of the characters and words are made.

It is often helpful to view the character being edited against a background that consists of a higher resolution version of the same character. The background image then provides a picture of what you want, as a guide for face editing. These background characters are usually created by scan converting the splines at a larger font size (in micas or points).

NOTE

If you want to use the background feature, you should generate another *Chars* file containing the same characters as the font that you want to edit, but in a bigger size. If the font being edited is 12 point, for example, the background characters might be 48 point; the resolutions of the foreground and background should be the same. The size of the background font must be an integral multiple of the size of the font being edited. Furthermore, the (integral) ratio of these two sizes must evenly divide the Dot size (see below). See subsection 6.5 for a further discussion of resolutions and sizes.

It is possible to edit with the same file as both the foreground and background, or, more generally, with another font of the same size.

6.1 Starting Edit from the main menu

To edit a font, proceed as follows:

a. Move the cursor to the block labeled "Edit" and depress any button on the mouse. If you use the *left* mouse button, the name "ACTemp" will appear in the Source file block, and the Background block will contain nothing. If instead you use the *right* button, the name of the last *Chars* file you used (other than ACTemp) will appear in the Source File block, and "ACTemp" will appear in the Background block.

b. Change, if necessary, the name in the Source file block to be the name of the *Chars* file that you would like to Edit.

c. Enter in the Background block the name of the *Chars* file that you would like to use as a background, (if any). If you don't want the background feature the Background block should be changed, if necessary, to contain nothing (Blanks and tabs do not qualify as "nothing"!!!). To do this, select the Background block and type DEL followed by RETURN.

d. Next, you should set the Dot size; this parameter determines how large a square of the Alto screen will be devoted to each dot in the character raster when that character is displayed for editing. It is desirable to have this size be as large as possible, subject to space and divisibility constraints.

   Typically, a dot size of 4 will be small enough to display an entire 12 point 384 bit per inch resolution character in the editing area of the screen. Other typical dot sizes are as follows:

Point Size	Dot Size
10	6
8	8
6	10

   Subsection 6.5 discusses the calculation of maximum dot size.

   The dot size must be an integer multiple of the ratio of background mica size to foreground mica size (Normally this ratio will be exactly the same as the ratio between their respective point sizes.) with a tolerance of plus or minus one mica in the background mica size. If not, the background image will not be displayed and a message will appear stating, "Scream: Background enlargement does not divide cell size."

e. Move the cursor to the block labeled "DO IT" and depress any button on the mouse. The main menu will be replaced by a new menu that we will call the *edit menu*.

   The edit command menu consists of three major parts:

   1) A large square in which the character to be edited is displayed.
   2) A menu of available commands to the right of the square.
   3) A rectangular area (below the square) where sample strings of text can be displayed.

It may be helpful at this point to refer to Fig. 6 1 at the end of this section for an illustration of the Edit menu. Please note that this illustration differs from the real menu in that on the real menu the labels without blocks around them (i.e., "**Symbol**," "**Octal**," "**File Character**," and "**Edit Character**") are actually displayed as white on black.

6.2 Filing Characters in Edit

There are three blocks in the section of Edit menu labeled "File Character".

a. Move the cursor to the block that says "Get," depress the left button on the mouse, and type in the character to be edited. The character will be displayed in the "Symbol" block, in the editing area, and in the small square in the command area.

NOTE

A message appears at the top of the screen telling you that the character was read from the "original file." Had this been a character which you had altered and filed during the current session of Edit, the character would have been read from a file called "ACEdits." More about ACEdits later.

If a character is too big to be displayed in the editing area of the screen at the chosen dot size, a message stating "Warning: tick mark for displaying widths lies off screen" will *usually* appear in the message area of the screen. In this case you should probably "Quit", pick a smaller dot size, and re-enter Edit.

The abscence of the warning message should not be taken as a guarantee that you have chosen a dot size small enough for displaying the entire raster, but only that the dot size is small enough for displaying the current foreground character’s widths. An example of this is a large capital "Q", which is taller than it is wide and which extends below the baseline. Parts of the character’s raster bitmap could fall above and below the display window and yet it is conceivable you would get no warning message. If you proceed to alter the bitmap (or the widths), file the character away, and then "QUIT", you will lose those parts of the raster which would not fit on the display.

You are probably safe, however, provided that the entire raster seems to be there (not an exclamation mark with a missing dot, etc.) and there is at least one column of empty grid squares constituting either a left-hand or right-hand border, and at least one row of empty squares constituting either a top or bottom border (empty *fractional* squares at the top and on the right don’t count).

NOTE

If the "Get" command is invoked with the *middle* mouse button instead of the *left* button, the desired character may be displayed by typing in the octal number. The *right* button automatically "Gets" the last character (if any) on the "ACEdits" file, which is described below.

b. Edit the the raster bits of the character using the mouse buttons. The mouse buttons perform the following functions:

Left Button	-	Turns the bit being pointed at by the cursor from white to black.
Right Button	-	Turns the bit being pointed at by the cursor from black to white.
Middle Button	-	Turns the bit being pointed at by the cursor to white if it is black and to black if it is white.

By depressing and holding the left or right buttons, you may slowly paint across an area and change all of the bits pointed at by the cursor to black (if left button is used) or white (if right button is used).

Each time you press a mouse button, a message appears at the top of the screen telling you the coordinates of the mouse within the display window, and whether you are drawing, flipping, or erasing bits. When you release the mouse button, a message will appear telling you the new coordinates of the mouse and the "Extent" in x and y directions from the point where you depressed the button. This feature is intended to aid measurements of character stroke thickness and character width.

As you make changes to the bitmap in the large square, the same changes are simultaneously recorded in the smaller square to the right. The position of the cursor is also tracked in the smaller square, but if this feature bothers you, you may turn it off by moving the cursor dead-center in the smaller square and depressing any button. Doing so a second time turns it back on.

c. As soon as a change has been made to the character, the label in the "Get" box becomes "Put," and the label in the "Delete" box becomes "Cancel." No other character may be displayed until the changes to the currently-displayed character are either "Put" or "Canceled".

d. When you have edited the bits of the character to your satisfaction, you may store the fruits of your labor. Invoking the "Put" command writes the new version of the raster into a file called "ACEdits" (for .AC Edits). When you

later invoke the "Quit" command to exit from Edit, the new characters described in the ACEdits file are merged into the original .AC file, overwriting whatever was there before.

e. At the completion of a "Put" operation, the label in the "Put" block changes back to "Get", and the label in the "Cancel" box changes to "Unput". At this point you have the option of either Getting another character or Unputting and hence Getting an earlier version of the same character. Unput requires confirmation unless you invoke it with the *right* mouse button. To confirm type "Y", "y", or RETURN for yes, or "N", "n", or DEL for no. If you invoke Unput with the *middle* mouse button, the earlier version will always be read from the *original* file (if it is there). If you invoke Unput with either the *left* or *right* mouse buttons, the *most recent prior version* of the character will be used, whether on the ACEdits file or on the original *Chars* file. Use "Unput" sparingly and carefully.

f. If in paragraph d. above you were dissatisfied with your fruits you could have chosen "Cancel" instead of "Put". With Cancel, no writing or erasing is done to the ACEdits file. The editing area on the screen is erased and an automatic Get is done on the character. Cancel requires confirmation unless you invoke it with the *right* mouse button.

g. The "Automatic" command is used to automatically *Put* the displayed character (if it has been altered) and automatically *Get* one of the adjacent characters in the input file. If Automatic is invoked with the *left* mouse button, PREPRESS starts counting octal numbers beginning with [the octal code displayed + 1] mod 400_8 and continues until it finds a character in the either the ACEdits, foreground, or background files, and *Gets* this character. In other words, invoking Automatic with the *left* mouse button causes the *next* character in the font (in terms of octal codes) to be displayed and similarly if the *right* button is used, the *previous* character will be displayed. The *middle* button will *Put* and *Get* the same character displayed. "Automatic" with the left mouse button is an extremely convenient way of skimming through an entire font.

h. Under certain rare circumstances you may wish to remove a character from a font completely. This may be done by using the "Delete" command. To avoid confusion, use Delete only on a character which has not yet been altered during the current editing session. Since "Delete" is destructive, irreversible (once you "Quit"), and rather unusual it requires two confirmations, unless you invoke it with the *right* mouse button, in which case it requires one confirmation. Do not do any other editing on this character lest this also lead

to confusing results.

6.3 The "Edit Character" section of the Edit menu

There are five blocks in the section of the Edit menu labeled "Edit Character".

a. A grid is provided to aid in the editing. To eliminate the grid, invoke the "Grid" command. To re-display the grid, select the "Grid" command a second time.

b. If a background file is being used, the background display may be alternately turned on or off by invoking the "Bkgrnd" block with any mouse button.

c. If the character being edited and the background character are not quite aligned, invoke the "Shift" command. Two locations must now be entered with the mouse. Move the cursor to a point on the background and depress any button on the mouse. Then move the cursor to a point where the specified point is to be moved and again depress a mouse button.

NOTE

The two points actually specify only relative motion and the two points specified can be anywhere inside the character display block as long as the relative distance and direction between the two points specified are correct.

d. The "Area" command computes the area of both the foreground and background characters and displays the results in the number of dots with respect to the foreground. The Area command can be used to help determine whether the current foreground character has more black area or less black area than the background character.

e. As a result of the editing, the height and width of the character may have been changed. New height and width information can now be entered with the use of the four tick marks which are displayed immediately outside the borders of the character drawing area. The marks are short straight lines which touch the border. The tick marks on the *left* and *bottom*, if extended, locate the origin of *both the foreground and the background characters*, i.e., the 0,0 point of the x and y coordinates of the characters. The origin is the place on the baseline where the character begins. The tick marks on the *top* and *right* define the end of the *foreground character's* width vector, rounding the characters actual width vector coordinates to the nearest grid spacing (The width vector of a character is the position that will be occupied by the origin of the next character in a string.). For a normal character in a non-rotated font, the left and right tick marks will line up, indicating that the y-component

of the *foreground* character's width vector is zero, but the bottom tick mark will be to the left of the top tick mark, indicating that the x-component of the *foreground* character's width vector has a positive value.

To reposition a tick mark, invoke the "Widths" command and then move the cursor until it is pointing at the mark to be moved and depress and hold any button on the mouse. The mark will begin flashing on and off. The mark will now follow the cursor as it is moved until the mouse button is released. Note that repositioning either the left or the bottom tick mark will cause the background to be redrawn when you leave the Widths command mode because you have tampered with the background character's origin point. Also keep in mind that moving any tick mark truncates the width vector to the nearest grid mark in both directions when the character is "Put" on the ACEdits file. To leave "Widths" command mode, move the mouse to the block labeled "Widths" or invoke any other command.

There are also two tiny square dots displayed in the border with the tick marks, one dot on the top and one on the right. If no background is currently displayed, these dots will be in an undefined position; often, but not always, they will be located immediately opposite the bottom and left tick marks, respectively. If a background is displayed, however, the dots will represent instead the end of the current *background character's* width vector. This width is actually comprised of floating point values (as in the case of the foregound), but is rounded to the nearest background, rather than foreground, grid spacing. These dots are helpful as reference points in setting the widths of the edited character.

NOTE

Location of the tick marks is very important for the proper positioning of each character. The tick marks not only specify the space needed for each character, but also the empty space between characters. If the empty space is incorrectly specified, the printed characters may not be evenly spaced. Some may look crowded together and others may look too spread apart. In addition, very small (6 to 8 points) and very large (greater than 14 points) sizes of the same characters may require non-proportional spacing. This occurs because additional spacing is required between some letters when they are very small to enhance readability and to eliminate a crowded look. This is an aesthetic judgement that must be made during this face editing process.

The empty space allotted for each character is usually evenly divided, with half of the space placed on each side of the character. The effects of repositioning the tick marks can immediately be

examined by viewing it in a string with other characters in the "letter-fit window."

6.4 Showing strings of characters

The large, long block near the bottom of the screen is called the "show-string box" or "letter-fit window."

a. At any time during face editing, the current character may be viewed between two "H" characters. This is a useful standard for checking letter fit when adjusting the width information of a character. To display the character between two "H"s, move the cursor into the show-string box and press the *left* or the *middle* button on the mouse. If any changes have been made to the current character which have not yet been "Put," a Put is done automatically. Alternatively, three copies of the character may be viewed by moving the cursor into the show-string box and pressing the *right* button on the mouse.

b. A more versatile way of displaying characters in the show-string box is by invoking the "Type in" command. If the command is invoked with the *left* mouse button, the user may type in a string of characters to be shown. Terminate with RETURN. If any changes have been made to the character currently in the editing area which have not yet been "Put," and this character appears in the string typed in, then a Put is done automatically. As much of the string as will fit into the box will be shown. You may type in the octal code for a character instead of typing the character itself by doing the following: type "#", followed by a 1-3 digit octal number, then hold down the Swat key and type "x", the octal code will be replaced by the appropriate character. If the *middle* mouse button were used to invoke the command and the string were too long to fit in the box, the user would have been asked if the remainder of the string were to be displayed. PREPRESS remembers the previous string that was displayed and if "Type in" is invoked with the *right* mouse button, the previous string will be displayed with updated characters.

c. The "Consec" command is used to display a string consisting of a consecutive sequence of characters. This is especially useful for comparing "stroke" thicknesses. Invoke the "Consec" command with the *left* mouse button, type RETURN, and a string will be displayed beginning with the currently selected character. If you invoke the "Consec" command with the *left* mouse button and type in a character (other than RETURN), a string will be displayed beginning with the typed character. If any changes have been made to the character currently in the editing area which have not yet been "Put," a Put is done automatically. Invoking the command with the *middle* button allows more characters in the source file to be displayed. The *right* button

automatically shows a string starting with the character previously specified with the Consec command.

d. Invoking the "Quit" command of the Edit menu will cause all of the new rasters sitting in the file ACEdits to be merged into the font being edited, with new versions of characters overwriting old, and will then return you to the main PREPRESS command menu. Quit requires confirmation unless you invoke it with the *right* mouse button. To confirm type "Y", "y", or RETURN for yes, or "N", "n", or DEL for no. If any changes have been made to the character currently in the editing area which have not yet been "Put," a Put is done automatically.

NOTE

Throughout the entire editing session the original *Chars* file is never altered, thus if you desire to flush the entire Edit session and get back to ground zero, you can always type <left-SHIFT><swat> or (groan) boot the Alto. Unless you delete the "ACEdits" file you can give yourself one last chance to change your mind if you re-enter Edit with the same file parameters. Read on:

By the way, one more thing: when the Edit command starts up, it looks around to see if there is an ACEdits file already in existence. If so, it offers you the option of merging those changes into the current *Chars* file before you start any further editing, if you would like.

6.5 Calculating dot size for Edit

To calculate the maximum dot size and the best background resolution for displaying in the PREPRESS Edit mode, proceed as follows:

a. Determine the maximum number of character dots to be displayed. This can be calculated by using the following formula:

$$\frac{\text{Point Size}}{72} \times \text{Resolution} = \text{Dots per Em Quad}$$

For example, a 10-point font and a printer resolution of 300 would require:

$$\frac{10}{72} \times 300 = 41.7 \text{ dots}$$

NOTE

The point size of the font is a safe number to use for this calculation since most characters fit within a square defined by the fiducials (on the em quad). However, if any character is known to be wider than the fiducial, or if all characters are known to be

smaller than the fiducial, the actual measurement in points of the largest character can be used.

b. Determine the maximum dot size for the display. This can be calculated by dividing the display area of 400 Alto bits by the number of required character dots obtained from step a. above. For the same 10-point font in the above example, the size would be:

$$\frac{400 \text{ bits}}{42 \text{ dots}} = 9.5 \text{ bits per dot}$$

or a maximum dot size of 9 (an integer). Since it is not desired that the dot size be an odd number, the next lower even number (8 in this case) would be used.

NOTE

The number 9 is the maximum dot size that can be displayed and it is possible to use the odd number if the *background* is obtained from the same file as the foreground, or if a 9 to 3 ratio is tolerable. The preference for even numbers comes only from the necessity of scaling the background by some factor to match the size of the foreground, and from the minimum dot size of 2 for a half-toned background. In general, even numbers have a greater variety of non-trival factorizations than do odd numbers.

c. Choose the size ratio to use for the background conversion. For the continuing example of the 10-point font, a good ratio would be 8 to 2.

d. Determine the background point size. This is calculated by multiplying the foreground point size by the point size ratio obtained in step c above. For the 10-point font this is:

$$10 \text{ points} \times (8/2) = 40 \text{ points}$$

So the .SD file should be reconverted at 40 points into the ACTemp file.

Typical foreground/background combinations for a 300 line per inch system would be as follows:

Foreground Point Size	Background Point Size	Dot Size	No Background Dot Size
8	40	10	11
10	40	8	9
12	48	8	8
14	42	6	6
16	32	4	5
18	36	4	5

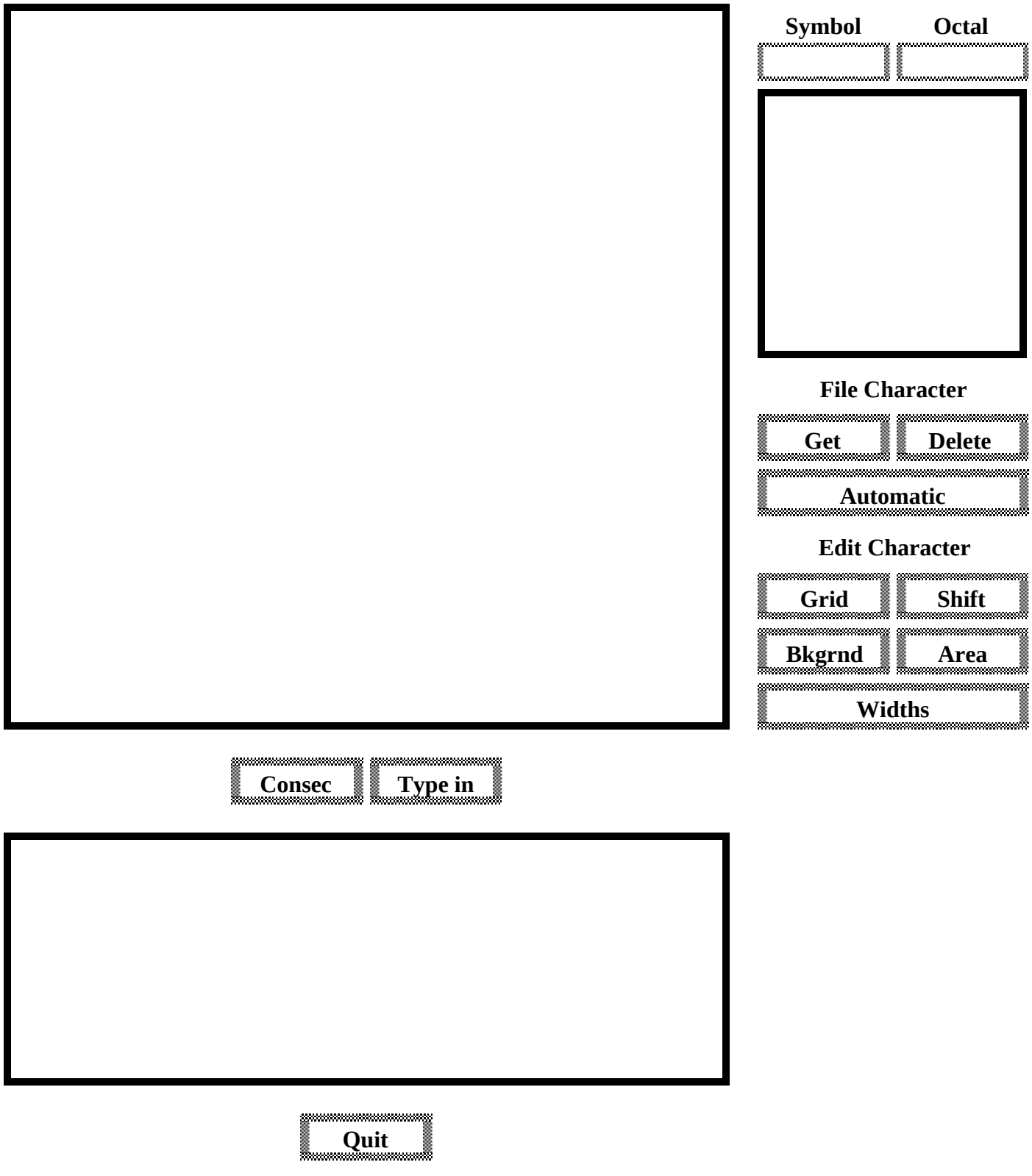

Figure 6-1: Prepress Edit Menu

7 Rotate: Rotating rasters

The "Rotate" block in the main menu can be used to rotate scan converted characters by any multiple of 90 degrees. Rotate will only work on input rasters of type *Chars*. To use this feature proceed as follows:

a. Move the cursor to the "Rotate" block of main menu and depress any button on the mouse. The names which appear in the relevant file blocks depend upon which mouse button you use, as described in Section 2 of this document.

b. Rotate expects the name of a *Chars* file to be entered in both the "Source" and "Output" file blocks, although the name in the Output file block need not be an existing file. The names may be changed if desired by moving the cursor to the block whose name is to be changed, depressing any button on the mouse, typing in the desired Source and/or Output file names, and terminate by depressing RETURN.

c. The block labelled "Angle" specifies the number of degrees through which the rasters should be rotated. This rotation will be in addition to whatever rotation the font already has. The angle to rotate is a signed decimal number in units of degrees (not minutes); positive means counterclockwise and negative means clockwise. The angle must be a multiple of 90. The default Angle is +90, but you may change this parameter if you wish by bugging the "Angle" block. The *left* button on the mouse adds 90 degrees to "Angle", the *middle* button adds 180, and the *right* button adds 270. If the resulting sum is greater than 180 degrees, then 360 degrees is subtracted. Rotation by 0 degrees, or by 360 degrees is an expensive no-op.

d. Move the cursor to block labeled "DO IT" and depress any button on the mouse.

8 Width: extracting the width vectors of a font

The "Width" block of the main menu is used to extract width information from a *Splines* or *Chars* font, and to output this information as a PrePress file with a single *Widths* segment. Dictionaries of these *Widths* segments can be used by formatting programs as sources of information about the sizes of characters; the file Fonts.Widths is just such a dictionary.

If you extract the width vectors from a *Splines* font, the resulting *Widths* segment will include information that can be scaled and transformed to determine the widths of any size and rotation of the font. Such *Widths* segments are distinguished by the fact that they carry a size of 0 points (which is 0 micas). This proportional scaling of

width information works well over a fair range of sizes, but can cause problems with very large or very small version of the font; small fonts really need slightly more space between characters than simple geometric scaling provides (or equivalently, large fonts need less than proportional space).

On the other hand, if you extract the width vectors from a *Chars* font, the resulting *Widths* segment will only describe the widths in micas of that particular size and rotation: no scaling or rotating will be performed. This type of *Widths* segment can be distinguished by the fact that they have an explicit nonzero size.

To use the Width command, proceed as follows:

a. Move the cursor to the "Width" block and depress any button on the mouse. The names which appear in the relevant file blocks depend upon which mouse button you use, as described in Section 2 of this document.

b. Enter name of the file from which width information is to be extracted in the "Source file" block (SDTemp, ACTemp, or other appropriate file) and terminate with RETURN.

c. When all of the information is entered, move the cursor to the "DO IT" block and depress any button on the mouse.

9 Rename: changing the attributes of a font

You sometimes want to change the index information of a single segment file in PrePress format. This need frequently arises because the ReadAL, ReadKS, and ReadCU commands fill in incorrect index information. But the need can also arise for other reasons: you might want to change the family name of a font, for example. The Rename command takes a single segment PrePress font file as input. When you enter the name of that file in the Source file block, the Family Name, Face, Size, Rotation, and (if appropriate) resolutions of the font will be displayed. You may alter any of these parameters. Then, when you select the "DO IT" block, the index information in the source file is changed to the new values that you have specified.

Note that the Rename command in PREPRESS and the Rename command to the Alto Executive are not even remotely similar. The Exec Rename command changes the name of a file, but leaves the data intact; the PREPRESS Rename command changes the header information inside the file and lease the filename intact.

When invoking Rename, the File Type block will display the type of the single segment of the input file: *Chars*, *OrbitChars*, or whatever, just as always. But the Rename command does not allow you to change this. To change the Type of the source file, you should use the appropriate PREPRESS command that really does the work: just calling a *Splines* segment a *Chars* segment doesn't make it one.

The Rename command alters the input file in place: there is no separate output file.

10 Coordinate: checking rasters against splines

This command is provided for helping to "coordinate" a hand-made or retouched font set with the spline "masters." For some applications, it is essential that the actual font symbols used are no larger than the scaled spline symbols. The "Coordinate" command helps to uncover the differences.

Note: The "Coordinate" command *will not work* from the main menu in Prepress1.13. The following instructions in the remainder of this paragraph are included anyway in the hope that this problem will be fixed in a later version of PREPRESS. "Coordinate" does, however, work if invoked from a command line. To invoke this command, move cursor to the block labeled "Coordinate" and depress any mouse button, then invoke the "DO IT" block. The names which appear in the relevant file blocks depend upon which mouse button you use, as described in Section 2 of this document. The Coordinate command needs: (1) the name of a *Splines* file (extension .SD) and (2) the name of a *Chars* file (extension .AC). (File (2) is usually the result of face editing operations with the Edit command.) The size and resolution parameters of the .AC file are crucial to the calculations used to decide whether the font is coordinated.

The results of the "Coordinate" command are written on a special text file named PrePress.Lst; any old version of PrePress.Lst, if any, is overwritten. For each character, the following are given:

a. The character width, followed by "s/b" and the spline width, converted into the resolution of the device.

   If the spline width is smaller than the character width, the warning [width] is printed as well. If the font is fully coordinated, all character widths will be less than or equal to the spline widths.

b. Coordinates of the edges of the bounding box of the black portions of the character, referred to the (0,0) point.

   The coordinates have the labels L, R, B, and T (for left, right, bottom, and top). Each coordinate prints the corresponding dimension for the character and then for the spline (scaled to the resolution of the device). If the font is fully coordinated, the box surrounding the character black will lie inside the box surrounding the spline black.

11 Grow and Shrink: adjusting the blackness of rasters

The Grow and Shrink commands provide a means for adding bits to and subtracting bits from each raster in a *Chars* file in order to compensate for the effects of positive and negative xerography. Growing a raster by *k* units turns black every raster dot that, in the original raster, was at distance no more than *k* from a black dot, where

distance is measured as the sum of the *x* and *y* distances. Shrinking a raster only leaves black those dots that, in the original raster, were at the center of black balls of radius *k* (where balls are also determined by *x* distance plus *y* distance). To use these commands proceed as follows:

a. Move the cursor to block labeled "Grow" (if it is desired to add bits) or to "Shrink" (if it is desired to remove bits) and depress any mouse button. The names which appear in the relevant file blocks depend upon which mouse button you use, as described in Section 2 of this document.

b. The name of a *Chars* file automatically appears in the Source file and Output file blocks. Either or both may be changed, if desired. Terminate with RETURN.

c. Enter in the block labeled "Bit Factor" the number of units you want to add (or subtract) from each character in the file and terminate with RETURN; the default is 1.

d. Move the cursor to block labeled "DO IT" and depress any mouse button.

12 Scale: altering the resolution of a raster

The Scale command allows you to translate in as much as is possible between rasters (that is, *Chars* files) of different resolutions. If you move from low resolution to high resolution, each dot of the input raster will generate many dots of the output raster; if you go the other way, an area of the input raster will get to vote about each dot of the output raster. This voting is controlled by a percentage, which defaults to 50%; if more than this percentage of the voting dots are black, then the resulting output dot will be black.

If you are attempting to build a reasonable set of low resolution rasters for a font, you will probably be revolted by the result of simply scan-converting splines at the desired resolution. Before giving up on the splines completely, you should try scan-converting the splines at a higher resolution (say, 400x400 if the font you want is 100x100), and then using Scale to reduce the resolution of the resulting rasters. This two step process often produces a better looking result. But don’t get your hopes too high: as a rule of thumb, all Alto-size fonts must be face edited by hand to look any good.

The Scale command needs a Source *Chars* file, and an Output file, whose names are set in the usual way. In addition, the Percent block contains the percentage used to decide the results of the voting discussed above. Finally, the desired resolution of the new raster is determined by the blocks labelled "X factor" and "Y factor"; these factors are floating point numbers which, when multiplied by the resolution of the

input raster, determine the resolution of the output raster. Thus, factors smaller than 1.0 imply moving to a lower resolution. Setting the X factor also sets the Y factor, since one normally operates with the same resolutions in both directions; if you want the X and Y factors set differently, set X first, and then Y.

CAUTION

The "Scale" command may cause baseline misalignment.

13 OrbitFormat and DeOrbitize: back and forth between *Chars* and *OrbitChars*

Recall that there are two different formats for a segment of a PrePress file that contains rasters: *Chars* and *OrbitChars* (see section 1). The OrbitFormat command takes a *Chars* font and produces the corresponding *OrbitChars* one, while the DeOrbitize command goes in the opposite direction. You should set the Source file and Output file in the normal way.

14 MakeAL and ReadAL: back and forth between *Chars* and .AL files

As mentioned briefly in section 1, an .AL file contains a font designed for use on the Alto screen in a format optimized for use with the Convert machine instruction on the Alto. The MakeAL command takes a *Chars* font and converts it into the .AL format. In order to correspond roughly to actual dimensions on the Alto screen, the resolution of the *Chars* font should be in the neighborhood of 72. The rotation of the *Chars* font should be zero.

A file in .AL format doesn’t include most of the attribute information that is carried along by files in PrePress format. Hence, the resolution, family name, and the like of the *Chars* input file are ignored by MakeAL: only the rasters themselves really matter.

The ReadAL command attempts to go in the reverse direction: it takes an .AL file as input, and constructs a *Chars* file with the same rasters. Since the .AL file doesn’t include attributes like family name, the ReadAL command cannot guarantee to get these attributes correct. ReadAL guesses the font attributes by looking at the name of the input file. If your input file is named "Helvetica10bi.AL", the ReadAL command will assume that these rasters form a 10-point HELVETICA with a BIR face. When ReadAL returns, you should invoke Rename on the resulting *Chars* file, and correct any wrong attributes.

Since a file in .AL format does not have as much information as a *Chars* file, going from a *Chars* to .AL and back to *Chars* will *not* get you back to the same place: the baselines, widths, and attributes of the final *Chars* file will only be approximations to those of the initial *Chars* file. However, the other loop is safe. If you go from .AL to *Chars* to .AL, you should get back to where you started. This makes the ReadAL

command particularly useful when you wish to add a few characters to an .AL font: just invoke ReadAL, Edit the resulting *Chars* file, and then MakeAL again.

The .AL format has some trouble with characters that overhang, unfortunately; here is the story in sad detail: Normally, one expects the black portions of a character to be located from left to right somewhere between that character's origin, and the origin prescribed for the next character. But some characters violate this assumption. A raster is said to *overhang to the left* if it includes black bits to the left of its origin; it is said to *overhang to the right* if it includes black bits to the right of the origin of the next character. Overhanging to the left is not too common, but overhangs to the right are very common: in particular, many fonts incorporate accents as characters with zero (or, if BRAVO is involved, very small positive) width that overprint the following character by overhanging to the right.

The MakeAL command adjusts for overhangs in the following manner: The .AL format can't handle overhangs to the left at all. Hence, if any character overhangs to the left, then the entire font is shifted to the right be just far enough so that no character overhangs to the left. Subsequent to this shift, MakeAL checks to see if any characters overhang to the right. The .AL format allows a character to overhang to the right as long as the overhang does not extend pat the next even multiple of sixteen, that is, as long as the length of the scan-line measured in words isn't increased by the overhang. Overhangs that don't violate this guideline will work correctly. MakeAL takes care of overhangs that exceed this guideline by one of two methods, depending upon the setting of the "Clip" flag: if the Clip flag is set, then that part of any overhang that extends into the next word will be omitted. If the Clip flag is false (the default value), then the width of the offending character is artificially extended by just enough to demand the right number of extra words, and the entire overhang is included in the raster.

The real solution to the overhanging rasters problem is to use the new "kerned strike" of .KS format discussed in the next section.

15 MakeKS, ReadKS, and MakeStrike: back and forth between *Chars* and the various flavors of strike formats

More recent Alto systems don't use the Convert instruction (which is not to be confused with the "Convert" command in PREPRESS) to put characters onto the screen, but instead use BitBlt. The various flavors of Strike files contain rasters for an Alto font in a format optimized for use with BitBlt. Files in the original Strike format are given the extension .STRIKE. The MakeStrike command will take a *Chars* font, and produce the corresponding .STRIKE file. Once again, you probably want the *Chars* font to have a resolution of 72, and to have rotation zero.

The original Strike format cannot handle characters that overhang in either direction, unfortunately; for the definition of "overhang", see the preceding section.

If you call MakeStrike on a font with characters that overhang to the left, it will bomb out. But, if your only overhangs are to the right, PREPRESS gives you two choices for what will happen, depending upon the value of the "Clip" flag. If the "Clip" flag is false (the default value), the width vectors of any overhanging characters will be artificially extended just enough to eliminate any overhang. On the other hand, if the "Clip" flag is true, then the overhanging bits will simply be ignored.

The real cure for the overhang problem is to use a new font format called KernedStrike format; files in this format conventionally have the extension .KS. For the details of .KS format, seethe memo "Font Representations and Formats" mentioned is SubSection **1.6**.

The command MakeKS will go from a *Chars* file to a .KS file, and the command ReadKS will go in the reverse direction. As in the case of MakeAL and ReadAL, be warned that the .KS format does not have as much header information as the .AC format; hence, the ReadKS command is forced to do some guessing about font attributes. After ReadKS is done, you should examine the attributes of the resulting .AC file and use Rename to change them if they are incorrect.

16 MakeCU and ReadCU: back and forth between *Chars* and .CU files

"Carnegie-Mellon" format, called .CU format, was once the standard format for raster fonts. It has the great virtue of simplicity, but is rather bulky and carries less attribute information than a *Chars* file.

The "MakeCU" command takes as input a *Chars* font, and produces a font in .CU format. Set the Source and Output files in the standard way.

The "ReadCU" command takes as input a .CU file, and produces a corresponding *Chars* file. As with ReadAL, the ReadCU command attempts to deduce from the file name and other heuristics the correct attribute information for the *Chars* file; after ReadCU returns, you should invoke Rename, and correct any wrong data.

Since a file in .CU format does not have as much information as a *Chars* file, going from a *Chars* to .CU and back to *Chars* will *not* get you back to the same place: the baselines, widths, and attributes of the final *Chars* file will only be approximations to those of the initial *Chars* file. However, the other loop is safe. If you go from .CU to *Chars* to .CU, you should get back to where you started.

17 ImposeWidths: resetting the widths of a raster font

The "ImposeWidths" command takes a *Widths* file as input, and overwrites the width information in the Output file with the widths given in the input. The Output file must be of type *Splines*, *Chars*, or *OrbitChars*.

This facility was added so that width information suitable for phototypesetter equipment could be imposed on a font, and that proof copies could be printed using Xerox standard printing facilities.

18 ReadWidths: constructing a *Widths* segment from scratch

The "ReadWidths" command reads an input text file in its own funny format, and uses the data in that input file to construct a *Widths* file. This just provides you with an easy way to construct an arbitrary *Widths* segment; these arbitrary widths can then be imposed upon an existing font with the ImposeWidths command.

The input file consists of a number of nouns followed by values:

NAME	font name
SIZE	point size
FACE	face code (e.g. BIR)
XL	xl value
YB	yb value
XW	xw value
YH	yh value
SCALE	number
WIDTHS	(see below)

The XL, YB, XW, and YH values are the dimensions of the font bounding box (see the document entitled Font Representations and Formats mentioned in Subsection 1.6).

The WIDTHS noun is followed by (character,width) pairs, terminated with the noun STOP. A character may either be the individual character or its octal code (a number with two or more digits). All the numbers given will be multiplied by the global SCALE factor (a floating-point number). For example:

```
A 104
72 230
STOP
```

If the widths are to be all the same, the noun ALL followed by the value will give the widths. Here is a sample file for the HyType printer:

```
Name HyType
```

Size 12
XL 0 YB -70 XW 254 YH 353
Widths
    All 254
    Stop

19 Comments about Dictionaries

The commands described in the remaining Sections are used to build and manipulate dictionaries that contain spline encodings, scan-converted character encodings, width encodings, or combinations of these. Recall that a *dictionary* is simply a file in PREPRESS format with (possibly) more than one segment. For example, the Width command is used to build *Width* files, each of which describes a single font. These tables can then be welded into a dictionary of width information with the commands described in the following Sections. Dictionaries have three uses:

a. Hold information in one place, permitting easy backup of fonts.

b. Deliver to printing programs the *Splines*, *Chars*, or *OrbitChars* segments required to actually print a document.

c. Deliver to formatting programs the *Widths* segments required to provide width information.

The dictionary commands generally operate by shuttling information between (typically small) Source and Output files, and a (perhaps huge) Dictionary file. By default, PREPRESS assumes that the Dictionary file contains information of only one type ("SD" is the default name of the spline encoding dictionary, "CD" that of the character encoding dictionary, and "WD" that of the width table dictionary). However, the name of the Dictionary file may be provided to any of these commands by invoking the "Dictionary" block with any mouse button and typing in the desired dictionary name.

Since dictionaries may contain various kinds of information, it is helpful to be able to tell the information apart. Spline encodings are always stored with size=0, to indicate that they may be scaled; character encodings always have non-zero size. (Internally, the different kinds of information are kept separate, but this separation does not always surface at the command level.)

20 Trident: using a trident disk

PREPRESS has facilities for dealing with files on a Trident disk, should your Alto be so equipped. In particular, on any of the upcoming Dictionary commands, you may, if you wish, set the "Trident" flag. The default value of the Trident flag is "Ignore", if you are using the command line, and "Prefer" if you are using the menu. "Ignore" orders PREPRESS to deal only with the normal Diablo disk system. If the Trident flag to is set to "Prefer", PREPRESS will look for *all* files (except Prepress.lst) first on the Trident, and failing that, on the Diablo.

NOTE

Due to the current state of the Dorado microcode, it is not possible

> to use the Trident connected to a Dorado both as a simulated Diablo disk and as a simulated Alto Trident. Therefore, when running on a Dorado, the value of the Trident flag is irrelevant, and all files reside on the standard system disk (one partition of the Trident simulating dual Diablo Model 44's).

If you are using a Trident disk, you should specify its drive number in the appropriate block. The drive number of a T-80 disk will be an integer between 0 and 7 inclusive. The larger T-300 disks include up to three logically distinct file systems, which are referred to in the standard way: for example, the three file systems on T-300 drive number 3 are called 3, 403, and 1003.

The Trident feature is usually used to allow the larger font dictionaries to reside on a large, fast disk. But note that PREPRESS tries for all files on the Trident first, not just Dictionary files. This can cause some confusion when combined with the following fact: Only the dictionary commands can talk to the Trident; the other commands (a different overlay) only talk to the Diablo. Suppose, then, that you scan-convert some splines into a desirable raster font, and you call the output file ACTemp; of course, this means ACTemp on the Diablo. Now, you want to Merge this raster font into a dictionary file located on the Trident. If the Trident file system should happen to contain a file named ACTemp, you are in trouble, since that file will be taken in preference to the one that you just created. Moral: never allow the Trident and Diablo file systems to share names (except Prepress.scratch).

Recall that PREPRESS uses the file PrePress.Scratch to keep temporary information in, during the execution of some commands. The above conventions about Tridents and Diablos apply to this file as well. Suppose that we are about to do a command involving a Dictionary, and that the Trident flag is set. If a PrePress.Scratch exists on the Trident, that file will be used; if none exists on the Trident, but there is one on the Diablo, it will be used; and if none exists either place, one will be opened on the Diablo. Some of the dictionary commands (such as a non-Fast Merge) need a scratch file as large as the dictionary on which they are working; hence, if you are playing with a large dictionary on the Trident, you had better arrange that a file named PrePress.Scratch exist on the Trident as well. It doesn't matter what is in this PrePress.Scratch file or how long it is; but it must exist (use Neptune to transfer your User.cm to the Trident under the name PrePress.Scratch if nothing else).

By the way, when working with Tridents, the program TFU (for Trident File Utility) is often helpful. This program handles such mundane issues as copying and renaming files on the Trident. The TFU "address" command can also check to see whether or not a file occupies a contiguous block of disk pages on the Trident; this is important to font wizards, because the font dictionary used by the PRESS printing server must be contiguous.

21 List: getting a dictionary's table of contents

This command lists some fraction of the contents of a dictionary file on the list file PrePress.Lst. Enter the dictionary's name in the Dictionary block, and set the Trident flag if necessary. The original contents of the file PrePress.Lst (if any) are overwritten. PrePress.Lst is always on the local (Diablo 31) disk.

If the Fast flag is set to true, the listing will describe each segment of the dictionary by a couple of lines of text which give its type and its attribute information. If the Fast flag is set to false, the listing will contain a summary for the data on every character in every segment as well; warning, a non-Fast listing of a large dictionary can be quite long.

In a non-Fast listing, different kinds of per-character information are provided for the different kinds of file segments:

a. Spline Encoding

   Width in x (fraction of point size); width in y; left edge of black information relative to (0,0) point; bottom edge; right edge; and top edge.

b. Character Encoding

   Width in x (bits); width in y; offset from (0,0) point to the left-most black of the character; offset in y to the bottom of the character; width of black; and height of black.

c. Widths

   Widths in micas. If size=0, scaling of widths is permitted and the number given should be multiplied by the size (in micas) and divided by 1000 to obtain the width in micas.

d. Tex Metrics

   Nothing at the moment.

When first using PREPRESS, it is probably advisable to do quite a lot of Listing. Note that List can handle any file in PrePress format, including those with only one segment. Thus, you may also List *Chars* files, *Splines* files, *OrbitChars* files, what have you; just enter the name of the desired file in the Dictionary block.

List distinguishes between the three possible flavors of Chars segments: *Chars*, *OrbitChars*, and *MultiChars.* For *MultiChars* segments, List will print out the number of old width tables that are present; but it will not print out the data contained in the old width tables.

22 Fast and Slow editing of dictionaries

The commands Merge, Supersede, Delete, and Compact are the facilities that PREPRESS provides for editing dictionary files; future sections will discuss each command in detail. Each of these commands can be used in one of two modes, either Fast or Slow, depending upon the setting of the Fast flag; this section discusses the difference between Fast and Slow dictionary editing. In addition, if the operation is done in Slow mode, setting the Verify flag will enable extra checking as discussed in the following section.

A dictionary file consists of an index portion followed by a data portion. The index portion is usually rather small; it consists of a sequence of index entries, each of which either defines the numeric code associated with the string form of a family name in the dictionary, or describes a data segment located in the data portion of the dictionary. An index entry for a data segment points to that data segment by giving its length, and the address in the file where it begins.

The index portion of a dictionary can't have any holes in it; the next index entry must start immediately after the previous one ends. But the data portion of the file is less constrained. Programs that use dictionaries are supposed to use the index to determine what data segments are in a dictionary, and then find the data segments they want by following the pointers in the index entries. There is no requirement that the data segments come in any particular order; and there is no reason why there can't be gaps in the data portion of the file, regions that aren't part of any data segment.

In the simplest kind of dictionary, the data segments begin right after the the last index entry, they occur in the same order as their index entries, and they are tightly packed together. This kind of a dictionary is called *compact*, since it occupies as little disk space as possible for any dictionary containing those data segments.

If you use Merge, Supersede, or Delete in Slow mode (with the Fast flag set to false), what happens is this: the input files are read, and PREPRESS decides what data segments should go into the output file. PREPRESS then opens the temporary file Prepress.Scratch, and builds the output dictionary there. When the output has been entirely written on Prepress.Scratch, PREPRESS will notify you with a sprightly message at the top of your screen. PREPRESS then turns around and copies that data to the original dictionary file, notifying you when the copying is complete.

The slow method of operation has problems and advantages. Since the output dictionary is being completely rebuilt, it is produced in compact form, and hence doesn't have any wasted space. Since the output is first built in a scratch file, a Slow edit operation is also fairly safe: if your machine crashes at any random instant due

to a parity error, cosmic ray, or whatever, you won't loose your dictionary. Either the crash comes before Prepress.Scratch is completely built, in which case your input files haven't been changed yet, and you can just start over; or the crash comes while Prepress.Scratch is being copied back to the dictionary file, in which case Prepress.Scratch itself contains the valid output of the editing operation.

The problem, of course, is that Slow-mode operations aren't very speedy. In particular, if you are trying to add a single new data block to a large dictionary, a Slow merge will have to copy all of the data blocks in the big dictionary twice, basically to no avail. This becomes truly painful if you are building a large dictionary by doing many successive merges of this sort. As a cure for this problem, there is Fast mode.

In Fast mode, the edits are done to the dictionary file in place, and as few changes are made as possible. For example, suppose that you are adding one new segment to a large dictionary; a Fast Merge will add the new data segment to the end of the dictionary, extending the file. Of course, the Merge also has to add a new index entry to the index portion of the file. In order to make room for the new index entry, it might be necessary to take the first data segment in the dictionary and move it to a new location at the end of the file as well, to get it out of the way. Finally, the Merge rewrites the index portion of the dictionary to install the pointers to the new data blocks.

The primary advantage of the Fast mode is speed; but this speed is gained at some cost. First, the output dictionary won't be compact. A Fast Delete of a data segment, for example, just removes the index entry that points to the segment, and leaves that space as a hole. Also, the data segments won't be in the same order as the index entries that point to them.

Secondly, a Fast edit is a trifle dangerous. In particular, if your machine should happen to randomly crash while the new index portion is being written, your dictionary will be left in a useless state. The data segments will still be there, but the index portion of the file will consist of pieces of the old and new indices, smashed together. PREPRESS warns you when it starts writing the new index portion, and then informs you when it is finished; if your machine crashes during that time, you are out of luck. Therefore, it is advisable to keep some form of backup copy of your dictionary in some other file or files if you plan to edit it in Fast mode. If you are building a large dictionary by merging together a bunch of .AC files, on the other hand, you might as well use Fast mode: if disaster strikes, you can always just start again from the beginning.

In all of the dictionary commands that use the Fast flag, the default value is True if you are using the menu interface, and False if you are using the command line.

As of version 1.13, the index entries in every dictionary output by PREPRESS will be in alphabetical order, to make it easier to read the text files produced by List. Index entries to data segments appear ordered first by type, then by rotation, then by family name, then by size, then by face, and finally, if *Chars* type, by resolution. In addition, as of version 1.13, any index entries that define the string-code correspondence for family names that aren't used in the dictionary will be deleted.

23 Slow-Verify mode

Data segments of type *Chars, OrbitChars,* and *MultiChars* have an internal structure that is a miniature version of the structure of an entire dictionary file. At the beginning of these segments, there is a table of header information that specifies the dimensions and widths of each character in the font. Next, there is a table of file pointers that give, for each character code, the location of the corresponding raster block. And finally, there are the raster blocks themselves. Most font software always writes the individual raster blocks in character code order, and without leaving any gaps; that is, the font segment is *compact* at the character level. But there are exceptions. For example, the program Metafont on Maxc writes the raster blocks in random order.

Some font software, such as the printing program Spruce, will perform better if all of the individual font segments are compact; in Spruce's case, this improves performance by helping Spruce to access the disk in ascending order of virtual disk address. Thus, if you have produced rasters with Metafont, it is advantageous to be able to compactify the resulting dictionary at the character level as well as at the font level. This is purpose of the Verify flag. If the Verify flag is set for a dictionary command, the output dictionary is written one character (raster block) at a time rather than one font segment at a time. This is slow, but it guarantees that the output dictionary will be compact at the character level.

It doesn't make sense to have the both the Fast and Verify flags set to True, and the rule is that the Verify flag wins: if the Verify flag is True, then the dictionary operation will be done in Slow mode regardless of the setting of the Fast flag.

When doing a dictionary command in Verify mode, PREPRESS takes the opportunity to check various consistency conditions about the dictionary. For example, the height and width of the rsater for a character are stored in two places in the dictionary. If these don't agree, the dictionary is in error, and could cause Spruce to fall into Swat. If any heights or widths don't match, a Verify will detect the problem and give an error message. In addition, Verifying will guarantee that all empty characters have valid empty raster blocks.

Thus, Verifying a dictionary is a way of checking its health and putting it into a

standard form at the character level. But it is quite slow.

24 Compact: cleaning up a dictionary

The Compact command is provided in case you want to massage a dictionary without Merging in anything new or Deleting anything old. Any Compaction of a dictionary will be in Slow mode, and hence will rewrite the data segments contiguously and in the same order as the corresponding index entries. It wouldn't be a bad idea to Compact a dictionary after performing lots of Fast Merges, for example, to make the dictionary's use of file space more ordered and rational. A Compaction with the Verify flag set will also make each character-type data segment compact at the character level. This is both more expensive and less useful, but it should be done to certify the health of dictionaries that are being released for use on printing servers.

25 Merge: adding to or editing a dictionary

This command merges two PREPRESS files. The two input files are specified in the Dictionary and Source file blocks (remember to set the Trident flag if necessary). The Merge command takes each segment in the Source file, and adds it to the Dictionary file; if the Dictionary file already contains a segment of the same type and with the same attributes, the new Source file segment will overwrite it. The Merge is performed in either Fast, Slow, or Slow-Verified mode, as determined by the Fast and Verify flagss.

Remember that a PREPRESS file with a single segment is also a valid dictionary. Hence, if you want to put the three files File1.AC, File2.AC, and File3.AC into a dictionary, proceed as follows: First, retreat to Alto Executive, and rename File1.AC to be Dict.fonts. By this operation, you have created a dictionary with one of the right segments. Now, go back into PREPRESS, and Merge the Source file File2.AC into the Dictionary Dict.fonts; then Merge the Source file File3.AC into Dict.fonts; then List the file Dict.fonts just to be on the safe side. (You could just Merge the .AC files into each other without any initial renaming, but sticking to the file naming conventions is usually the best idea.)

See the next section for a discussion of what happens if either the Source or the Dictionary file contains *MultiChars* segments.

Note: If you bug "DO IT" with either the first or the second mouse button, and if the source file specified is not on your local disk or if it is not a single-segment Prepress file, Prepress requires confirmation before continuing.

26 ReviseWidths: merge, and update *MultiChars*

ReviseWidths is an option on the Merge command, but it deserves a Section unto itself. Unless you *really* know what you are doing, you probably don't want to set the

ReviseWidths flag. ReviseWidths is another facet of the *MultiChars* phenomenon, so, before reading further in this Section, you should reread Subsection 1.5.

There is one situation where a knowledgeable font wizard would use the ReviseWidths option: if he or she is putting a new version of a collection of raster fonts into the font dictionary of a printing server that runs the SPRUCE program. With the ReviseWidths flag set, what will happen is this: The Source file should consist of *OrbitChars* segments. If one of these segments represents a new font, not yet contained in the dictionary, it will simply be added to the dictionary. But, if an *OrbitChars* or *MultiChars* segment already exists for that font, a new *MultiChars* segment will be generated that has the new rasters and all of the widths, both the new and all of the old, appropriately timestamped (up to a maximum of 3 old width blocks are allowed).

In more detail, here is what happens if the two files that you are Merging both contain either a *MultiChars* or *OrbitChars* segment for a font. If the Source file has a *MultiChars* segment, then that segment simply replaces whatever was in the Dictionary. So suppose that the Source file has an *OrbitChars* segment. There are four cases left to consider, depending upon the type of segment in the Dictionary and the value of the ReviseWidths flag.

If ReviseWidths is true and the Dictionary segment is type *OrbitChars*, then the two *OrbitChars* blocks from the Source and the Dictionary are combined into one *MultiChars* segment in the output file. This *MultiChars* segment has the rasters from the Source segment; it has the widths from the Source segment labelled as expiring in the infinite future, and the old widths of the Dictionary segment labelled as expiring at the instant of the Merge.

If ReviseWidths is true and the Dictionary segment is type *MultiChars*, the output will be another *MultiChars* segment with rasters from the Source as the only rasters, the widths from the Source as the current widths, and the old widths from the Dictionary pushed down one level.

If ReviseWidths is false and the Dictionary segment is *OrbitChars*, then the Source segment replaces the Dictionary segment in the usual way.

If ReviseWidths is false and the Dictionary segment is *MultiChars*, then the most recent rasters and widths in that *MultiChars* segment are replaced by the rasters and widths from the Source file, while the old widths are left unchanged.

27 Supersede: merge, but handle *Widths* specially

This command is identical to Merge except in the processing of *Widths* segments. Recall that there are two different types of *Widths* segments in the world. Those that

came from *Splines* segments contain width information that can be scaled to handle any size and rotation; those that came from *Chars* segments only handle one size and rotation. For now, let us call the first type *S-Widths* and the second *C-Widths*; (remember that you can distinguish them because an *S-Widths* segment has size zero).

The Supersede command (called, in earlier versions of PREPRESS, the Supercede command) allows you to clean up a dictionary of *C-Widths* segments if you are now lucky enough to have a *S-Widths* segment. In particular, if there is an *S-Widths* segment in the Source file, it will be added to the Dictionary file, and all existing *Widths* segment of either variety that include subsidiary information will be deleted.

Using *S-Widths* segments has the advantage that the width dictionaries can be much shorter. You only need one *S-Widths* segment for each family and face, whereas you need one *C-Widths* segment for each family, face, size, and rotation. If a width dictionary has both *S-Widths* and *C-Widths* segments for the same font, formatting programs will preferentially use the *C-Widths*. Thus, you can make a particular size and rotation of a font space non-proportionally if you desire.

Note: If you bug "DO IT" with either the first or the second mouse button, and if the source file specified is not on your local disk or if it is not a single-segment Prepress *Widths* file, Prepress requires confirmation before continuing.

28 Delete: removing a segment from a dictionary

This command is used to delete a font from a dictionary. Enter the dictionary's name in the Dictionary block, and set the Trident flag if necessary. The rest of the input information is a collection of attributes that are used to specify the proper segment to delete. The following types of information must be specified:

a. Family name
b. Face type
c. Size
e. Rotation
f. Resolutions

You can't avoid specifying these things, since they each default to something. You should be able to specify the File type as well, but you can't in this version of PREPRESS.

By specifying a size of zero, you are requesting a segment that is either of type *Splines*, or is a *Widths* segment that came from *Splines*. In this case, the specified size, rotation, and resolution are ignored.

If no segment exists in the dictionary with the correct attributes, nothing is deleted

and a complaining message appears. Delete can be run in either Fast, Slow, or Slow-Verified mode; in Fast mode, only the index to the dictionary is affected by the Delete

29 Extract: copying a segment from a dictionary

The purpose of Extract is to pull a single segment of interest out of a dictionary. Enter the dictionary's name and deal with the Trident issue as usual. Then, by specifying attributes exactly as in the Delete command, identify the particular segment of interest. The Extract command will read the dictionary file to find the segment of interest, and then write a single segment PREPRESS file that contains that segment. The dictionary file is not changed. The name of the resulting single segment file is determined by the Output file block. If no segment exists that has the correct attributes, a complaining message appears.

Be warned that attempting to Extract a *MultiChars* segment from a dictionary has exotic effects; in particular, the segment is translated into an *OrbitChars* segment by throwing away all of the old widths as a side effect of the Extract.

Appendix A: Using the command line

As commented in Section 2, the PREPRESS program can be driven either from the command line or from the menu. All of the above discussion assumed that you were using the menu. But there are situations where the command line approach would be more convenient. The basic command line format is:

PrePress.Run <com>/s <arg>/s <arg>/s <arg>/s . . . <arg>/s.

The command name is specified in <com>, and it may be followed by switches. Arguments usually require switches. All command input may be in upper or lower case. Command names may be abbreviated as long as they remain unambiguous.

When using the command line, it is more trouble to specify file names; hence, the default names that PREPRESS assumes for convenience when driven from the menu assume a more mandatory nature when you work from the comand line.

Finally, here is a list of the PREPRESS commands with a description of the appropriate arguments, switches and default file names.

ReadSF: Unswitched argument is .SF files to read. Output is to the file SDTemp. Set the Update flag with the /U switch on the command. Set the Incline by an argument with the /I switch (example: 20/I). Set the X and Y factors by arguments with the /X and /Y switches (example: 1.25/X). The I, X, and Y arguments apply to all of the .SF files following them on the command line.

Convert: Source file is SDTemp; output file is ACTemp. Thicken with the /T switch to the command. Set size with /P switch for points, /M for micas. Set both resolutions with the /D switch (unit is 10 times the number of dots per inch); set just the Y resolution with /E (same unit). Set rotation with /R (unit is degrees, not minutes). Set italic slant with /I.

Show: Source file is ACTemp, but you can specify another *Chars* file as an argument with /S.

Edit: Source file is ACTemp, but you can specify another *Chars* file as an argument with /S. Specify the background *Chars* file (if you want one) by giving its name as an <arg> with the /B switch. Dot size is 20 unless you set it to something else as an argument with the /D switch.

Rotate: Source and output are ACTemp. Set the amount of rotation to be added by an argument with the /A switch; the argument will be interpreted as a signed decimal number of degrees (not minutes), and must be a multiple of 90.

Width: Output is to WDtemp. Specify ACTemp as the input by giving the /C switch to the command, or specify SDTemp as the input by giving the /S switch to the

command.

Rename: Source file is set by the command switch: /C for ACTemp, /S for SCtemp, and /W for WDtemp. New attributes are specified as arguments. Use /N for family name, /F for face, /P for size in points, /M for size in micas, /R for rotation (in degrees), /D and /E for resolution (according to conventions for Convert).

Coordinate: You will be prompted for the file names. You may override the size and resolution data in the *Chars* input file by giving a size and/or resolution as arguments; same conventions as Convert.

Grow and **Shrink**: Source and output are ACTemp. Set the Bit factor with /D.

Scale: Source and output are ACTemp. Set the X and Y factors with /X and /Y; setting X also sets Y for convenience. Set the Percent with /D.

OrbitFormat, **DeOrbitize**: Source and output are ACTemp.

MakeAL, **MakeCU**, **MakeKS**, and **MakeStrike**: Set Source with /S, set Output with /O; if unspecified, the Source defaults to ACtemp.

ReadAL, **ReadCU, ReadKS**: Set the Source with an unswitched argument; the output is to ACTemp.

ImposeWidths: Source file (for widths) is WDtemp. Set the output file with command switch: /C says ACTemp, /S says SDTemp.

ReadWidths: Set the Source with an unswitched argument; the output is to WDtemp.

List: Put /F after the command to set the Fast flag; put /T *before* the command (on the word PrePress) to go to the Trident. A command switch of /S, /C, or /W sets the Dictionary file to SD, CD, or WD respectively. Alternatively, you can give the Dictionary file as an argument with the /B switch.

Compact: Put /V after the command to set the Verify flag; put /T on the word PrePress to go to the Trident. The command switches /S, /C, and /W set the Dictionary default to SD, CD, and WD respectively. The Dictionary default can be overridden by an argument with the /B switch.

Merge, **Supersede**: Fast and Trident flags are set as in List. Verify flag is set as in Compact. The command switches /S, /C, and /W set the Source and Dictionary files to SDTemp and SD, ACTemp and CD, and WDtemp and WD respectively. The Dictionary default can be overridden by an argument with the /B switch.

Delete: Fast and Trident flags are set as in List, Verify flag as in Compact. Dictionary name is specified by a command switch or argument/B as in Merge.

Attributes are set as in Rename.

Extract: Output and Dictionary file names are set via command switches and arguments exactly like the Source and Dictionary names in Merge. Attributes are set as in Rename.

A.1 A command line example

Suppose it is desired to build a dictionary of font files for use by a Dover printer. It is desired to include spline definitions for Helvetica and Times Roman, along with scan-converted versions for quick reference: 10 point Helvetica, and 10 and 12 point Times Roman. The dictionary is to be called G.

```
PrePress ReadSF HELVETICA.*-SF          (Read Helvetica splines)
Rename G _ SDTemp
PrePress Convert 10/P 3840/D        (Make 10-point font at 384 bpi.)
PrePress OrbitFormat          (Orbitize ACTemp.)
PrePress Merge/FC G/B           (Fast Merge ACTemp into G.)
PrePress ReadSF TIMESROMAN.*-SF          (Read Times Roman into SDTemp.)
PrePress Merge/FS G/B           (Merge SDTemp into G.)
PrePress Convert 10/P 3840/D        (Make 10-point font at 384 bpi.)
PrePress OrbitFormat          (Orbitize ACTemp.)
PrePress Merge/FC G/B           (Merge ACTemp into G.)
PrePress Convert 12/P 3840/D        (Make 12-point font at 384 bpi.)
PrePress OrbitFormat          (Orbitize ACTemp.)
PrePress Merge/FC G/B           (Merge ACTemp into G.)
PrePress Compact G/B        (Compact the dictionary to clean up)
PrePress List/F G/B           (Get summary listing to verify G.)
Empress 6/p PrePress.Lst          (Print the summary listing in 6 point.)
```

Appendix B: Prepress revision history

Prepress version 2.1:

Modified September 28, 1980 10:20 PM by Lyle Ramshaw:
// Changed TEX face encoding/decoding to work backwards.
// Fixed MapACtemp to follow the pointer in the IX to find
// the data segment in the input instead of assuming that
// that the data segment immediately follows the EndIX.
// Added the Verify flag to the MergeDelete commands:
// a Verified operation sorts the individual raster blocks
// within Chars, OrbitChars, and MultiChars fonts, and checks that
// the CharWidth blocks and raster blocks give consistent
// descriptions of the raster dimensions. Fixed several
// bugs in the Area command of FEdit. Result is 2.1.

//Modified September 17, 1980 12:10 PM by Kerry LaPrade:
// Edit function now works differently for Dot Size of 1.
// Also in Edit, fixed bug so that characters which do
// not fit in the editing area are once again clipped
// more properly. Updated files: AuxiliaryMenuBox.bcpl,
// FEdit.bcpl, FEditFile.bcpl, FEditUtil.bcpl, and
// Prepress.bcpl. Result is 1.21.

//Modified September 12, 1980 5:48 PM by Lyle Ramshaw:
// changed storage allocation in MergeDelete to allow larger
// dictionaries to be merged. Also went through and made
// most of the static variables "page zero statics" to save
// space; the statics themselves don’t take up as much room,
// and the code that references them is shorter and runs
// faster. Result is 1.20.

//Modified August 18, 1980 9:56 PM by Lyle Ramshaw:
// fixed bug in MakeStrike: the Length field in the StrikeBody was
// getting computed wrong. (How is it that this wasn’t noticed
// before, I wonder..) Result is version 1.19.

//Modified July 31, 1980 11:40 AM by Lyle Ramshaw:
// fixed bug in PrepressMenu1; adjusted dictionary commands
// so that all family names are output in all caps, and with all
// all trailing bytes = 0. Result is version 1.18.

//Modified July 6, 1980 9:53 PM by Lyle Ramshaw:
// Delete was bitten by the same CompareIx problem that bugged
// Extract last October!! So I moved the Extract patch to a
// place where it fixes both problems. In passing, I also added

// a "Beginning dictionary command..." message; result
// is version 1.17.

//Modified June 25, 1980 12:42 PM by Lyle Ramshaw:
// Improved the quadratic equation solver in SCVMain; result
// is version 1.16.

//Modified June 17, 1980 3:11 PM by Lyle Ramshaw:
// Fixed bug in Show command: released version 1.15.

//Modified June 15, 1980 3:33 PM by Lyle Ramshaw:
// Added a Junta back to levDisplay, to save enough space so
// that the Edit overlay will fit for a little while longer (sigh)...
// Also installed new Trident package.

Prepress version 1.13 (May 22, 1980 by Lyle Ramshaw, PARC)

This version will run on Dorados emulating Altos; on a Dorado, the floating point is done in software and the Trident disk stuff is disabled.

The code for editing dictionaries has been redone. The primary new feature is the Fast mode, in which the dictionary is rewritten in place. Minor new features are that the index now appears in sorted order, and that family names with no associated fonts are deleted. In passing, fixed a longstanding bug that sometimes smashed fonts when doing Merges into dictionaries that contained MultiChars blocks.

Added the new segment type TexMetrics for use by the local Tex hackers; PrePress doesn't create or modify such segments, but it will Merge, Extract, and Delete them from dictionaries.

Added the MakeKS and ReadKS commands that translate back and forth between .AC and the new kerned strike format, .KS. In the process, altered the MakeStrike command by changing the old "Kerned" flag to the new "Clip" flag. The MakeAL command also now pays attention to the "Clip" flag. "Clip" flag is settable from the command line with /K.

Changed the DecodeFace and EncodeFace definitions and calls to allow for a fourth letter in the face code (X, A, or O), and also to allow for numeric face codes (TEX logical sizes). For details, see the new version of "Font Representations and Formats."

Added stack space to the Trident overlay to prevent a fall into Swat when the requested file is not on the Trident.

Fixed a bug in the menu interface that caused falls into Swat under certain conditions, especially when several

commands are selected before any are executed.

Restored the FileName/B feature for driving the dictionary commands from the command line.

Changed List so that MultiChars segments are distinguished from OrbitChars, and took some of the carriage returns out of the List output to save space.

Changed the overlaying structure to make overlaying faster. The first time that the .RUN file is started, PrePress will go through the file to find the starts of all of the overlays, and store these in the .RUN file. Subsequent invocations of PrePress and overlay swaps within PrePress will be fast, working from a table of file addresses.

Prepress version 1.12 (incompatible with OS versions prior to OS 17)

Modified April 7, 1980 by Kerry LaPrade, XEOS and Lyle Ramshaw, PARC

Fixed bugs introduced in version 1.11 to Main Menu user interface and to writing of family names in dictionaries. Fixed MakeStrike to handle kerned characters more reasonably. Increased List capacity from 100 to 200 names, and changed carriage return-line feed sequences to simply carriage returns. Restored and modified command line capabilities for calling Edit, which first disappeared in version 1.9. Updated the set of device names which the /D and /E command line switches recognize. Changed command line spellings from Supercede to Supersede. Also, Supersede changed to again allow dictionary and/or Trident files as source files from the menu. Incorporated Trident microcode version of 1-31-80.

Prepress version 1.11 (incompatible with OS versions prior to OS 17)

Modified February 7, 1980 by Kerry A. LaPrade, XEOS

Sped up FEdit "Get" command for high resolution (small dot size) editing. Added obscure BravoX-like capability to "TypeIn" chars > 200b in FEdit show string box. Added toggleable "gnat" to update box. Increased size of show string box (again) necessitating transferal of some resident code to overlay in order to allow use of moderately-sized sysFonts.

Decreased size of resident code by moving some stuff from Prepress.Bcpl to PrepressCommands.Bcpl.

Cleaned up PrePressMenu code for smarter and more protective user interface. Different mouse buttons do different things. In general, first and second mouse buttons invoke standard default, and third button reverts to last thing used. As per Lyle Ramshaw’s sugestions and instructions, fixed Trident drive menu specification to allow access to multiple directories on T-300’s. Menu boxes can no longer be bugged in the wrong order. User is warned if source file isn’t on local (sysDisk) disk.

Fixed list bug for chars > 200b.

Lyle Ramshaw (PARC) changed representation of blank characters in .AC files as produced in OrbitFormat.bcpl, and bounding box calculation as produced in Convert.bcpl.

Prepress documentation is now kept as [MAXC]<AltoSource>PrepressDocSources.DM.

Changed ReadAL to create .AC files at 72 bpi instead of 77 bpi.

Added measurement messages to bit manipulations in Edit, and "ACEdits" vs original file notification to "Get" command in Edit. Changed background/foreground ratio determination to have a

plus or minus 5 mica tolerance in background size as opposed to plus or minus 1 previously.
Changed "Incline" prompt to percentage instead of degrees.

Modified December 21, 1979 by Lyle Ramshaw, PARC
Put in an OS version warning. Prepress 1.11 Trident and Edit sections are incompatible with OS versions earlier than OS 17.
Added a Notify window to the main menu, for prompt strings.
Installed the new Trident microcode of Ed Taft.
Fixed the Rename bug associated with ACTemp's parameters.
Installed the fix for the "ConvertAChar returns 14" bug (discussed below).
Allowed Tridents other than drive zero to be accessed.
Made ReadAL and DeOrbitize accessible from the command line.
Improved Rotate to handle rotations by any multiple of 90 degrees:
there is a new window in the main menu called "Angle" that holds the amount of rotation to be added, positive or negative, measured in degrees and stored in the static angleToRotate.

Modified December 5, 1979 (by LaPrade)
Changed user type-in scheme to be echoed inside menu box instead of dsp, reducing the amount of resident code and freeing up room for work on FEdit overlay. Enlarged FEdit "show string" box. Sped up FEdit "Put" command for high resolution (small dot size) editing. Updated .d files to David Boggs' 1979 versions. Miscellaneous housekeeping.

Modified November 26, 1979
A bug fix by Lyle Ramshaw in the microcoded floating point, and in the Convert routine, Arithmetic Splines. The microcoded version of FCM was using a signed instead of an unsigned compare to compare the lower order parts of the mantissa, and hence came up with the wrong answers once in a great while.

The Arithmetic Splines version of the routine SCVReadRuns had an apparent problem as well, which I fixed, although I never found an actual case where this bug showed up once the FCM bug mentioned above was fixed. The problem is the following: in the ArithmeticSplines version of SCVMain, each monotone spline segment is considered as linearly interpolating the spline values at the points t0, t0+dt, t0+2*dt, ... t0+K*dt=t1. The value of dt is calculated in floating point from the values of t0, t1, and K. But then, while finding intersections with successive scan lines, the t values are computed by repeated addition of dt, as in the sequence above. If rounding error leads to t0+K*dt being substantially different than t1, and if the spline has as s value at t1 a number just barely bigger than an integer, the s value at t0+j*dt might be smaller than that integer (which will be smax) for all j.

Prepress version 1.10 (incompatible with OS versions prior to OS 16)

Modified October 26, 1979
Bug fix by Lyle Ramshaw in Extract command, related to MultiChars hack.

Modified by Lyle Ramshaw, September/October 1979. The main changes were:
-getting a version of the Trident BR's compatible with OS17
-adding the ReadAL command, to go from .AL to .AC

-adding the DeOrbitize command, orbitized .AC to vanilla .AC
-loading a newer version of the microcoded floating point
-adding a window to the main menu which displays the file type

Prepress version 1.9

Modified September, 1978 by Kerry A. LaPrade, XEOS
Major improvements to "Edit" section:

- now uses Keith Knox's menu package
- business-like looking menu displays more information than before
- larger editting area
- less likely to accidentally change character's widths
- can turn background off during session
- confirmations required before invoking "hazardous" functions
- faster grid, exclusively ored for eraseability without leaving holes
- new window for viewing character being editted at one Alto pixel per printer pixel
- strings are clipped if necessary
- "Auto" command makes filing easier

Bob Sproull fixed a bug in ImposeWidths, and Joe Maleson revised the menu to allow calling ImposeWidths with arbitrary source and destination files.
File names CDTemp and CDEdits were replaced by ACTemp and ACEdits, respectively.