Go To

IPL Reference

Frame Technology Corporation
1010 Rincon Circle
San Jose, California 95131
USA

Frame Technology International Limited
Unit 52, Airways Industrial Estate
Cloghran, Dublin 17
Ireland

March 1993

Important Notice

Frame Technology® Corporation (Frame®) and its licensors retain all ownership rights to the FrameMaker® computer program and other computer programs offered by Frame (hereinafter collectively called “Frame Software”) and their documentation. Use of Frame Software is governed by the license agreement accompanying your original media. The Frame Software source code is a confidential trade secret of Frame. You may not attempt to decipher or decompile Frame Software or develop source code for Frame Software, or knowingly allow others to do so. You may not develop passwords or codes or otherwise enable the Save feature of Frame Software. Frame Software and its documentation may not be sublicensed and may not be transferred without the prior written consent of Frame.

Only you and your employees and consultants who have agreed to the above restrictions may use Frame Software (with the Save feature enabled), and only on the authorized equipment.

Your right to copy Frame Software and this publication is limited by copyright law. Making copies, adaptations, or compilation works (except copies of Frame Software for archival purposes or as an essential step in the utilization of the program in conjunction with the equipment), without prior written authorization of Frame, is prohibited by law and constitutes a punishable violation of the law.

FRAME TECHNOLOGY CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL FRAME BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OR DATA, INTERRUPTION OF BUSINESS, OR FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND, EVEN IF FRAME HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES ARISING FROM ANY DEFECT OR ERROR IN THIS PUBLICATION.

Frame may revise this publication from time to time without notice. Some states or jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

In the United States, Frame, the Frame logo, FrameMaker, FrameReader, Frame Technology, and FrameViewer are registered trademarks, and FrameBuilder, FrameMaker International Dictionaries, FrameMath, FrameServer, and FrameWriter are trademarks, of Frame Technology Corporation.

The following are trademarks or registered trademarks of Frame Technology Corporation in countries outside of the United States: Frame, the Frame logo, FrameBuilder, FrameMaker, FrameMaker International Dictionaries, FrameMath, FrameReader, FrameServer, Frame Technology, FrameViewer, and FrameWriter.

The spelling and thesaurus portions of Frame Software are based on THE PROXIMITY LINGUISTIC SYSTEM © 1992 Proximity Technology Inc.; C.A. Stromberg AB; Espasa-Calpe; Hachette; IDE a.s.; Kruger; Lluis de Yzaguirre i Maura; Merriam-Webster Inc.; Munksgaard Int. Publishers Ltd.; Nathan; Text & Satz Datentechnik; Van Dale Lexicographie bv; William Collins Sons & Co. Ltd.; Zanichelli. All rights reserved.

PANTONE® Computer Video simulation used in Frame Software may not match PANTONE-identified solid color standards. Use current PANTONE Color Reference Manuals for accurate color. PANTONE Color Computer Graphics © Pantone, Inc. 1986, 1988.

The following are trademarks or registered trademarks of their respective companies or organizations:

Apple, AppleTalk, Macintosh, LaserWriter, ImageWriter, Finder, MultiFinder, QuickDraw, MacroMaker / Apple Computer, Inc.

Proximity, Linguibase / Proximity Technology Inc.

Sun Microsystems, Sun Workstation, TOPS, NeWS, NeWSprint, OpenWindows, TypeScaler, SunView, SunOS, NFS, Sun-3, Sun-4, Sun386i, SPARC, SPARCstation / Sun Microsystems, Inc.

All other brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Any provision of Frame Software to the US Government is with “Restricted Rights” as follows: Use, duplication, or disclosure by the Government is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause at FAR 52.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement. Any provision of Frame Software documentation to the US Government is with Limited Rights. The contractor/manufacturer is Frame Technology Corporation, 1010 Rincon Circle, San Jose, CA 95131.

Go To

IPL Reference Contents

1 Introduction 1

This manual describes how to customize the SunView and X Window System versions of a Frame Technology® program to use a page description language other than PostScript. To customize a Frame product for a different page description language, you must perform the following basic tasks:

- Write a printer driver that converts a file in IPL (the Frame Intermediate Printer Language) into a page description file for a particular printer.
- Reconfigure a Frame product to use the correct font files and call your printer driver.
- Create font files that a Frame product uses to decide where to break lines and to display the results on the screen.

This manual gives you the information you need to complete these tasks. In Chapter 2, you'll find a complete description of IPL. Chapter 3 describes how to set up a Frame product to call your printer driver. Chapter 4 tells you how to add fonts to a Frame product.

> **Important:** This manual describes release 4.0 of IPL. In general, Frame Technology plans to keep future versions of IPL as compatible as possible with earlier versions. We can't guarantee, however, that future versions of IPL will be completely backward compatible.

2 Intermediate Printer Language 2

This chapter contains a description of IPL (the Frame Intermediate Printer Language). Here you'll find information on the IPL file format, its coordinate system, and each of its commands.

Introducing IPL

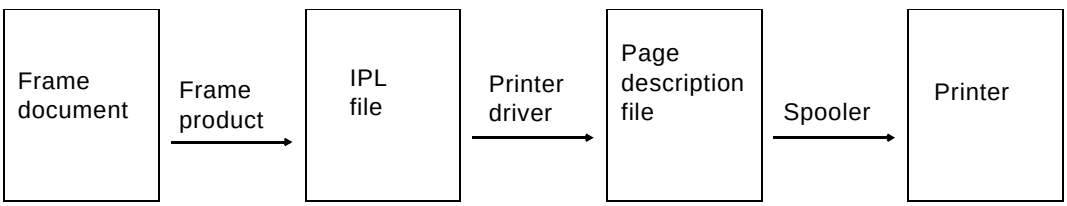
This section briefly introduces you to IPL. It includes Frame's goals and assumptions when designing IPL.

What is IPL?

IPL is a language for describing a Frame document in an ASCII file. An IPL file has a simple structure and a limited number of commands. It is designed to be read as data by a printer driver and translated into a page description file.

How a Frame product uses IPL

A Frame product creates an IPL file describing a document each time the document is printed. A Frame product printer driver reads the IPL file and translates it into a page description file. Finally, the system print spooler sends the page description file to the printer, where it is interpreted and printed.



The standard Frame product printer driver supports the PostScript page description language. Your printer driver will replace the PostScript driver. It will read the IPL file as data and create a page description file specific to your printer.

Design goals and assumptions

When designing IPL, Frame had several goals:

- Frame wanted IPL to be a relatively simple language. Because an IPL file is an ASCII file with one IPL command on each line, it is easy to read and print. A command's parameters are on the same line as the command—your program doesn't have to scan ahead for information when processing the file. Although this format makes the file less compact, file size usually isn't a problem because the IPL file is removed after translation.
- A Frame product is isolated from any particular page description language. In theory, different printer drivers can translate the same IPL file into page descriptions for different printers. For example, IPL statements contain CMYK (cyan, magenta, yellow, and black) and RGB (red, green, and blue) methods of describing color.
- Frame wanted to isolate the printer driver from the Frame product user interface. For example, when a Frame product user prints a range of pages instead of an entire document, the Frame product creates an IPL file containing only the pages the user selected.
- Frame wanted to isolate the programmer of a printer driver from the internal workings of a Frame product. An in-depth, technical knowledge of a Frame product isn't required to write a printer driver (although a working knowledge is helpful). For example, you don't need to know the Frame product document file format.

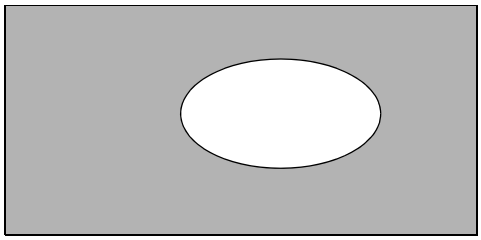
A Frame product, and thus IPL, expects an output device that uses a page description language with roughly the power of PostScript. In general, Frame assumes that, together, your driver and printer have most of the capabilities of the Frame product Tools window. They should be able to:

- Print with the various fonts that a Frame product requests
- Print a string of text at any location on the page, rotated in any direction (between 0.0 and 360.0 degrees) with specified interword and intercharacter spacing
- Draw outlined and filled graphics
- Define a clipping boundary within which text and graphics can appear. (For a complete definition of the clipping boundary, see "flip" on page 13.)
- Print CMYK separations for a document in document order or sorted by separation when using any printer. (Separations are black and white.)
- Allow definition of arbitrary colors in CMYK or RGB color systems
- Handle emulsion side up or down requests when printing to a device that requires this information
- Print or skip blank pages when printing color separations
- Print positive or negative images of document pages

- Draw objects on top of others and have the objects placed on top partially obscure those underneath

  For example, the printer should let you place a white oval on a black rectangle:

In general, the printer should allow you to place one object on another to obscure the object below.



- Print registration marks no matter what type of color separation is specified
- Print an imported graphics file at a particular location on a page, crop the image if part of it falls outside the clipping boundary, and partially obscure it if an object is placed on it

IPL file format

An IPL file has a very simple structure. It is an ASCII file with only one command on each line. Command names are always at the beginning of the line, and are terminated by a space (if parameters follow) or by a newline character. Parameters are separated by spaces. For example:

```
File identification ———— Maker Intermediate Print File 4.0
document command ———— document 576.00 396.00 1.00 1.00 1 1 0 0 1 0 0
State command ———— linewidth 0.50
beginpage command ———— beginpage 1 "1" "Right"
Comment ———— % Textrect Begin
             InvertText 0
             definefont 0 Times-Roman 3.00 12.00
             font 0
             text 72.00 81.00 0 0 26:This is a simple document.
Comment ———— % Textrect End
endpage command ———— endpage

endjob command ———— endjob
eof command ———— redefinefont 0 Times-Roman 1 2 2
(end of file)  eof 1 278
```

Important: Command names must use the case shown in this manual. The first line in the file is the file identification. Use this line to verify that the file is the right version and type.

Following the identification line is the `document` command. This command describes global information such as the size of the page and the initial values of some state settings.

One or more state commands can follow the `document` command. (For information about state commands, see "IPL states," next.) The example uses the `linewidth` state command.

Following the state commands are pairs of `beginpage` and `endpage` commands (one pair of commands for each page). Between the `beginpage/endpage` pairs are the commands defining the contents of the pages.

Lines beginning with a percent sign (%) are comments.

After the last `endpage` command is summary information (signalled by the `endjob` command). All fonts used in the IPL file are listed so your program can quickly identify them all. For more information about the summary, see "File structure commands" on page 6.

The last line in the file is always the `eof` (end-of-file) command.

IPL states

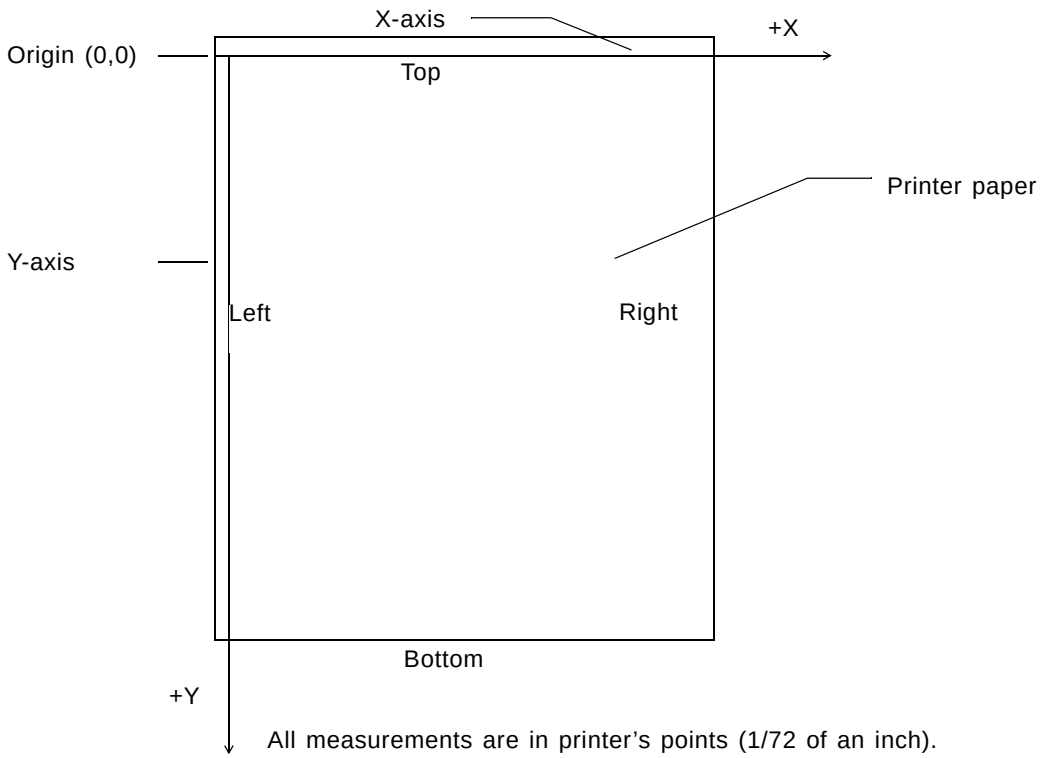
IPL has several state commands that set the value of parameters for the graphics drawing and text commands. The drawing and text commands use these values implicitly. For example, all text commands use the font specified in the most recent `font` command.

All states have an initial value that remains the current state until changed. State values apply across page boundaries.

For a complete description of each state command, see "State commands" on page12.

IPL coordinate system

Graphics and text are positioned using a coordinate system superimposed on the page as shown in the following illustration. All measurements are in printer's points (1/72 of an inch).



Command descriptions

In the following sections, you'll find a description of each IPL command presented in the following format:

CommandName

CommandName ParameterOne ParameterTwo ... ParameterN

Each parameter is one of the following types:

Integer	A base 10 number with no decimal point
Real	A base 10 number with a decimal point
String	Alphanumeric characters containing no spaces and not surrounded by quotation marks A string can contain a backslash (\) followed by the three-digit octal representation of the character's numeric code. (For more information, see "text" on page 18.)

CountString	An integer, a colon (:), and a string of alphanumeric characters. The integer gives the length of the character string. Each space and each three-digit character code (preceded by a backslash) counts as one character. The following three lines contain sample 10-character strings:
	`10:abcdefghij`
	`10:abc def gh`
	`10:abc \x80efghi`

Qstring	Alphanumeric characters surrounded by quotation marks (can contain spaces).
Boolean	The numbers 0 or 1, where 0=false and 1=true

Some commands have a syntax similar to the following:

*CommandName Number [ParmX ParmY]*Number

If a group of parameters appear in brackets as shown, the parameters must appear *Number* times in the order given. For example, if *Number* is 3, then there are three parameter pairs (*ParmX, ParmY*):

ParmX0 ParmY0 ParmX1 ParmY1 ParmX2 ParmY2

File structure commands

File structure commands describe document properties and identify components of the document file.

beginpage

beginpage *Landscape PageNumber PageName*

Landscape	Boolean
PageNumber	Integer
PageName	Qstring

The beginpage command marks the start of a group of commands describing a single page of a document. (Each page is ended by an endpage command.) The *Landscape* parameter indicates whether the page has a landscape or portrait orientation; a true value (1) indicates landscape orientation. The *PageNumber* and *PageName* parameters are generally for information only and aren't needed by most printer drivers. *PageNumber* is the number of the page in the IPL file. The first page in an IPL file is page 0. PageName is the Frame product description of the page (for example, if the page number style is roman, page number 4 will have a *PageName* of iv).

beginregistration

`beginregistration`

The `beginregistration` command and the `endregistration` command (see page 8) bracket the registration marks in the IPL file. These commands force the printer to print registration marks no matter what type of color separation has been specified.

colors

`colors` *`Number-of-colors`*

`Number-of-colors`	Integer

`Number-of-colors` is the count of how many spot color statements follow this statement. The IPL file needs to specify each `spot` color that you print (see page 11). This statement must come after the document statement and before the inkpalette statement in the IPL file.

document

`document` *`ScaleX ScaleY DocHeight DocWidth NumberCopies Collate LowResolution ManualFeed PrintDirection Negative Emulsion`*

`ScaleX`	Real
`ScaleY`	Real
`DocHeight`	Real
`DocWidth`	Real
`NumberCopies`	Integer
`Collate`	Boolean
`LowResolution`	Boolean
`ManualFeed`	Boolean
`PrintDirection`	Boolean
`Negative`	Boolean
`Emulsion`	Boolean

The `document` command contains several global variables determining the overall look of the document and how it is to be printed. *`ScaleX`* and *`ScaleY`* determine whether the document pages are to be scaled (typically they are set to 1.00, meaning a one-to-one ratio). For example, if both *`ScaleX`* and *`ScaleY`* equal 2.00, all the dimensions in the IPL file should be doubled. *`DocWidth`* and *`DocHeight`* describe the width and height of the document pages, respectively.

`NumberCopies` sets the number of copies to be printed. The value of *`Collate`* determines whether the document copies should be collated (a true value (1) means that copies will be collated). *`LowResolution`* indicates whether to print imported graphics in “draft” resolution or the printer’s highest resolution; a true value (1) means that images

should print in low resolution. *ManualFeed* determines whether pages should be manually fed into the printer; a true value (1) means that pages will be manually fed. *PrintDirection* sets whether the document is to be printed last page first or first page first; a true value (1) means the order of the pages in the IPL file is first page to last page.

Negative determines if white pages print as black and black pages print as white. A true value (1) means print negative pages; a false value (0) means print positive pages. If you are not printing color separations and you specify *Negative*, colors other than black and white are undefined.

Emulsion specifies if the printer prints with the emulsion side up or down when printing. A true value (1) prints pages with the emulsion side down; a false value (0) prints pages with the emulsion side up.

endjob

`endjob`

The `endjob` command signals the beginning of the summary. When your driver encounters this command, it should complete the page description file and exit. The `endjob` command is followed by `redefinefont` commands, listing all the fonts used in the document. (These commands are generally read only if your printer requires font information up front. For more information, see the `eof` command.)

endpage

`endpage`

The `endpage` command marks the end of a group of commands describing a page. When your printer driver encounters this command, it should execute the printer command that ejects a page from the printer.

endregistration

`endregistration`

The `endregistration` command and the `beginregistration` command (see page 7) bracket the registration marks in the IPL file. These commands force the printer to print registration marks no matter what type of color separation has been specified.

eof

`eof` *`NumberPages NumberFonts SummaryOffset`*

NumberPages	Integer
NumberFonts	Integer
SummaryOffset	Integer

The `eof` command is always the last command in the file. The *`NumberPages`* parameter is the total number of beginpage/endpage pairs in the IPL file. The *`NumberFonts`* parameter is the total number of fonts used in the IPL file. The *`SummaryOffset`* parameter is the byte number within the IPL file of the first statement of the summary. It occurs just after the `endjob` command.

If your printer requires all font information before reading the page descriptions, your driver can search to the end of the IPL file, go back 40 bytes, search forward for the `eof` command, and read the three parameters. Then it can search to the *`SummaryOffset`* location, and read all the `redefinefont` commands listing the fonts referred to in the IPL file.

inkpalette

`inkpalette 32`
`Pattern`$_1$
`Pattern`$_2$
`...`
`Pattern`$_n$

`Pattern` A gray shade (real), bit pattern (string), or no pattern (string)

The `inkpalette` command defines a document's fill and border patterns. Each pattern appears on a separate line following the `inkpalette` command. The patterns are numbered sequentially beginning with 0. The first 16 lines (patterns 0 to 15) correspond to the 16 Frame product fill and border patterns. A Frame product does not assign the second 16 patterns (patterns 16 to 31). However, your printer driver can use them. They normally contain rotated versions of the first 16 patterns to use for printing landscape-oriented pages. Frame product users can customize pen and fill patterns. For more information, see *Managing Frame Products*.

Each pattern is either a gray shade, bit pattern, or no pattern. A gray shade has the following format:

`G` *`GrayValue`*

where *`GrayValue`* is a real number that represents a shade of gray. Gray shades range from 0.00 to 1.00, where 0.00 is black and 1.00 is white.

A bit pattern has the following format:

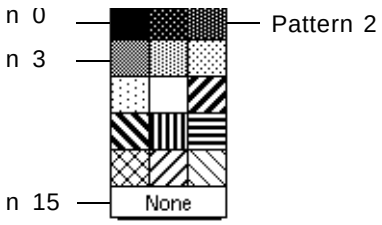
`P` *`BitPattern`*

where *`BitPattern`* is a string representing an ink pattern. Each pattern is defined using 16 hexadecimal (hex) digits. Two hex digits (8 bits) represent one row in the pattern; there are 8 rows.

The following pattern line represents no pattern (no fill or no pen):

`N`

For example, the following illustration shows the standard Frame product pen patterns (pen and fill patterns are always the same):



The `inkpalette` command for the standard Frame product patterns is:

```
inkpalette 32
G 0.00
G 0.10
G 0.30
G 0.50
G 0.70
G 0.90
G 0.97
G 1.00
P 0f1e3c78f0e1c387
P 0f87c3e1f0783c1e
P cccccccccccccccc
P ffff0000ffff0000
P 8142241818244281
P 03060c183060c081
P 8040201008040201
N
G 1.00
G 0.90
G 0.70
G 0.50
G 0.30
G 0.10
G 0.03
G 0.00
P f0e1c3870f1e3c78
P f0783c1e0f87c3e1
P 3333333333333333
P 0000ffff0000ffff
P 7ebddbe7e7dbbd7e
```

Gray values
Pattern 0: black
Pattern 7: white
Bit patterns
Pattern 15: no color

keepblankseps

`keepblankseps`

When printing separations, `keepblankseps` forces the printer to print pages with no images. Place this command before the first print page in your IPL file.

noseparations

`noseparations`

Use this statement when you want to print a composite page (a proof page).

skipblankseps

`skipblankseps`

When printing separations, `skipblankseps` forces the printer to omit pages with no images. Place this command before the first print page in your IPL file.

spotcolor

`spotcolor` *`Color-name CyanPercentage MagentaPercentage YellowPercentage BlackPercentage RedAmount BlueAmount GreenAmount`*

`Color-name`	CountString
`CyanPercentage`	Real
`MagentaPercentage`	Real
`YellowPercentage`	Real
`BlackPercentage`	Real
`RedAmount`	Real (0 to 255)
`GreenAmount`	Real (0 to 255)
`BlueAmount`	Real (0 to 255)

The specified color is printed as a spot color (not a process color). This implies there will be a separation for the spot color later in the IPL file (see "Sep" on page 15). The values for *`CyanPercentage,`* *`MagentaPercentage`*, *`YellowPercentage`*, and *`BlackPercentage`* are the percentages (0.00 to 100.00) of those colors, respectively, that make up the spot color. *`RedAmount, GreenAmount`*, and *`BlueAmount`* are arguments for the RGB color set. Each set must describe the same color. Both definitions are in the IPL file to let the printer determine which set it needs to use.You can specify any number of spot colors, provided the `colors` statement has the correct count for all spot colors in the IPL file (see page 7).

State commands

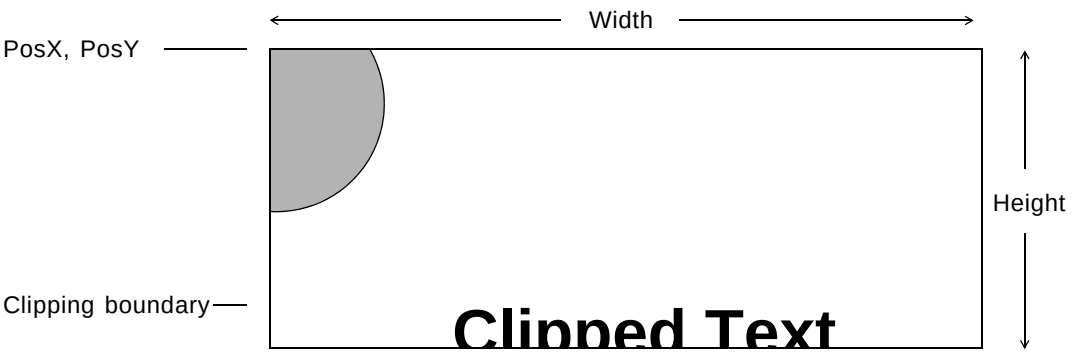
Clip

`Clip` *`PosX PosY Width Height`*

`PosX`	Real
`PosY`	Real
`Width`	Real
`Height`	Real

Initial value: the maximum imageable area of the printer.

The `Clip` command is a state command that sets the size of the clipping boundary by defining a rectangle. *`PosX`* and *`PosY`* are the coordinates of the upper-left corner of the rectangle. *`Width`* and *`Height`* are the dimensions (in printer's points) of the rectangle. All graphics and text outside the clipping boundary should be cropped at this boundary.



endoverprint

`endoverprint`

The `endoverprint` and `startoverprint` commands (see page 16) bracket objects which are overprinted.

Fill

`Fill` *`FillPattern`*

`FillPattern`	Integer

Initial value: undefined.

The `Fill` command is a state command that sets the current fill pattern used by all the drawing commands that require a fill. *`FillPattern`* is a number from 0 to 15. Fill patterns are defined by the `inkpalette` command (see page 9).

flip

`flip` *`FlipFlag`*

`FlipFlag`	Boolean

Initial value: undefined.

The `flip` command is a state command that controls whether or not objects are flipped left-to-right. It applies only to text and imported graphics. When *`FlipFlag`* is 1, objects are flipped; when it is 0, they are not flipped. Text is flipped around its reference point, and an imported graphic is flipped around its center. If objects are both flipped and rotated, they should be flipped first.

Forceps

`Forceps` *`ColorName CyanPercentage MagentaPercentage YellowPercentage BlackPercentage RedAmount GreenAmount BlueAmount`*

`ColorName`	Rendering Mode strings are `spot`, `process`, and `no`, depending on how your printer renders the color	CountString
`CyanPercentage`		Real (0.00 to 100.00)
`MagentaPercentage`		Real (0.00 to 100.00)
`YellowPercentage`		Real (0.00 to 100.00)
`BlackPercentage`		Real (0.00 to 100.00)
`RedAmount`		Real (0 to 255)
`GreenAmount`		Real (0 to 255)
`BlueAmount`		Real (0 to 255)

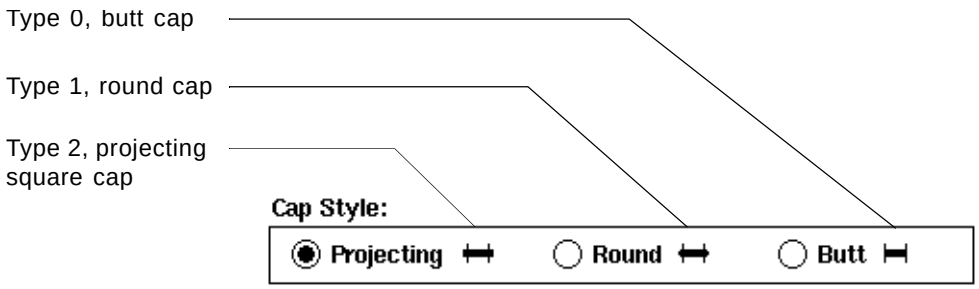
The `Forceps` command is a state command that is similar to the Sep command (page 15). However, `Forceps` uses PostScript to change the current separation color, even if the color in the current statement is the same as the previous color. A Frame product uses the `Forceps` command when switching in and out of color separation printing. *`CyanPercentage`*, *`MagentaPercentage`*, *`YellowPercentage`*, and *`BlackPercentage`* are the arguments for the CMYK color set. *`RedAmount`*, *`GreenAmount`*, and *`BlueAmount`* are arguments for the RGB color set. Each set must describe the same color. Both definitions are in the IPL file to let the printer determine which set it needs to use.

linecap

`linecap` *`CapType`*

`CapType`	Integer

The `linecap` command is a state command that sets the current line cap style. The *`CapType`* maps directly to the Cap Style properties in the Frame product Line Ends Options dialog box. When the CapType is 0, lines have butt caps; when it is 1, they have round caps; and when it is 2, they have square caps.



A square line cap continues beyond the endpoint of the path for a distance equal to half the line width and is squared off. A round line cap is a semicircular arc, with a diameter equal to the line width, drawn around the endpoint and filled in. A butt line cap is squared off at the endpoint of the path; it does not project beyond the end of the path.

linewidth

`linewidth` *`Width`*

`Width`	Real

Initial value: undefined.

The `linewidth` command is a state command that sets the current width of the border lines used by the graphics drawing commands. (For more information about the drawing commands, see "Graphics drawing commands" on page 20.) The *`Width`* parameter is the width of the line in points.

Pen

`Pen` *`PenPattern`*

`PenPattern`	Integer

Initial value: undefined.

The `Pen` command is a state command that sets the current pen pattern. The current pen pattern is used by most of the drawing commands. (For more information about the drawing commands, see "Graphics drawing commands" on page 20.) *`PenPattern`* is a number in the range 0 to 15. Pen patterns are defined by the `inkpalette` command (see page 9).

rotate

`rotate` *`Angle`*

`Angle` Real

Initial value: undefined.

The `rotate` command is a state command that specifies an object's angle of rotation. It applies to text columns, textlines, graphics. The *`Angle`* parameter is the angle of rotation (between 0.0 and 360.0 degrees) counterclockwise from the baseline of text or from the center of the graphic.

Sep

See the `Forceps` command.

separation

`separation` *`#:name`*

`#:name` CountString

`#:name` is a counted string of characters; it can be *cyan*, *magenta*, *yellow*, *black*, or a color specified in a `spotcolor` statement. It tells the printer to print the separation for the named color. The `separation` *`#:name`* command appears after a `beginpage` statement in the IPL file. (See the description of `CountString` on page 6.)

startoverprint

`startoverprint`

The `startoverprint` and `endoverprint` commands (see page 12) bracket objects which are overprinted.

Text commands

Text commands describe text to print.

comment

`comment` *`CommentText`*

`CommentText`	CountString

`CommentText` is a counted string of characters to print, generally in an alphanumeric representation. (See the description of `CountString` on page 6.) There are no extra spaces to the right of the colon; if a space appears here, it should be printed. However, characters outside the range 40 to 176 octal are represented in the form:

`\`*`nnn`*

where *`nnn`* is the three-digit octal representation of the character's numeric code. Most European characters and some common keyboard characters (like the backslash) use this three-digit octal representation. See also "text" on page 18.

Any spaces in the text string should be printed with the natural space width of the current font. (For information about the natural space width, see "definefont," next.)

definefont

`definefont` *`FontNumber FontName NaturalSpaceWidth FontSize StandardEncoding`*

`FontNumber`	Integer
`FontName`	String
`NaturalSpaceWidth`	Real
`FontSize`	Real
`StandardEncoding`	Boolean

The `definefont` command describes a font used in a document. There is only one `definefont` command for each font and it appears before the first use of the font in the IPL file. The *`FontNumber`* parameter is the number used throughout the document by the font command to set the current font. The number is suitable for use as an index into an array of fonts (from 0 to *n,* where the practical upper limit is 255).

The *`FontName`* is the name of the font as the Frame product understands it. *`NaturalSpaceWidth`* is the natural width of a space (in printer's points) of the font being

defined. *FontSize* is the size of the font (in printer's points). *StandardEncoding* determines if the font uses the standard Frame product character set. If the AFM file contains the following line,

```
EncodingScheme AdobeStandardEncoding
```

then the font uses standard encoding.

font

`font` *FontNumber*

FontNumber	Integer

The `font` command is a state command that sets the current font (the font that following text commands use). *FontNumber* is the reference number of a font that has been defined earlier in the IPL file with a `definefont` command.

InvertText

`InvertText` *InvertState*

InvertState	Boolean

The `InvertText` command is a state command governing whether text printed by the text commands is inverted. When *InvertState* has a true value (1), text is printed in white on a clear background; if inverted text doesn't have a dark object beneath it, the text is invisible. When *InvertState* has a false value (0), text is printed in black (on a monochrome printer) or the current separation color (on a color printer) on a clear background.

redefinefont

`redefinefont` *FontNumber FontName NaturalSpaceWidth FontSize StandardEncoding FirstPage LastPage*

FontNumber	Integer
FontName	String
NaturalSpaceWidth	Real
FontSize	Real
StandardEncoding	Boolean
FirstPage	Integer
LastPage	Integer

The `redefinefont` command's syntax is similar to the `definefont` command. (For information on the first five parameters, see "definefont" on page 16.) The *FirstPage* parameter is the number of the first page within the IPL file on which the font appears, and the *LastPage* parameter is the number of the last page on which the font

appears. (The first page in an IPL file is page 0.) There is one `redefinefont` command for each `definefont` command, repeating exactly the same font information.

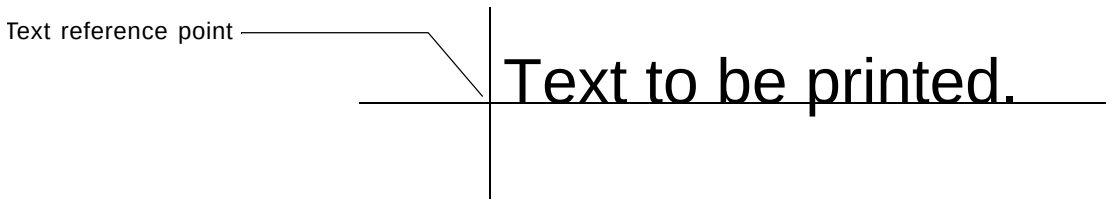
All the `redefinefont` commands appear in the summary after the `endjob` command. The purpose of this list of commands is to group all the fonts used in the IPL file. If the page description language requires you to set up font information first, your driver can skip to the first `redefinefont` command and identify all the fonts used in the IPL file. (For more information, see "eof" on page 8.) If the page description language doesn't require font information first, your driver will never see the `redefinefont` command because it needn't read past the `endjob` command.

text

`text` *PosX PosY TextToPrint*

PosX	Real
PosY	Real
TextToPrint	CountString

The `text` command is used to print text that is unjustified and has no extra space between characters. The *PosX* and *PosY* coordinates give the location of the text reference point (the point of the first character in the string). The text reference point is at the intersection of the baseline and the left-side bearing of the first character.



TextToPrint is a counted string of characters to print, generally in an alphanumeric representation. (See "CountString" on page 6.) There are no extra spaces to the right of the colon; if a space appears here, it should be printed. However, characters outside the range 40 to 176 hexadecimal are represented in the form:

nnn

where *nnn* is the three-digit octal representation of the character's numeric code. Most European characters and some common keyboard characters (like the backslash) use this three-digit hexadecimal representation.

Any spaces in the text string should be printed with the natural space width of the current font. (For information about the natural space width, see "definefont" on page 16.)

textB

`textB` *`PosX PosY Padding Spread TextToPrint`*

`PosX`	Real
`PosY`	Real
`Padding`	Real
`Spread`	Real
`TextToPrint`	CountString

The `textB` command appears when the text is fully justified and the characters are spread. It has both the *`Padding`* parameter of the textP command and the *`Spread`* parameter of the textS command. When your driver encounters a space in *`TextToPrint`*, it should add both the *`Padding`* and *`Spread`* values to the natural space width.

textP

`textP` *`PosX PosY Padding TextToPrint`*

`PosX`	Real
`PosY`	Real
`Padding`	Real
`TextToPrint`	CountString

The `textP` command is similar to the `text` command, but it has one additional parameter: *`Padding`*. This parameter is the space width (in printer's points) your driver should add to the natural space width each time it encounters a space in the text string. Generally, you see this command when the text is fully justified.

textS

`textS` *`PosX PosY Spread TextToPrint`*

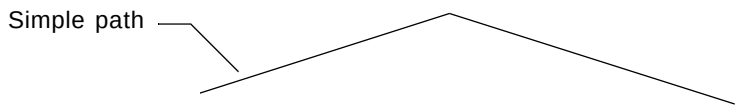
`PosX`	Real
`PosY`	Real
`Spread`	Real
`TextToPrint`	CountString

The `textS` command is similar to the `text` command, but it has one additional parameter: *`Spread`*. This parameter is the space (in printer's points) your driver should add between every pair of characters (even between a space and another character). This value maps directly to the Spread text box in the Character Designer and Paragraph Designer.

Graphics drawing commands

In IPL files, a graphics object is defined with a series of coordinate points. These points describe the path of the object (a series of connected line segments). The path, however, is a mathematical representation—it doesn't take into account the line width of the object's border. (For more information, see "linewidth" on page 14.)

For example, here is a simple path with three coordinate points (the coordinate points are at the path's endpoints and vertex):



Here is the polyline (with its border) defined by the path (the white line represents the path):

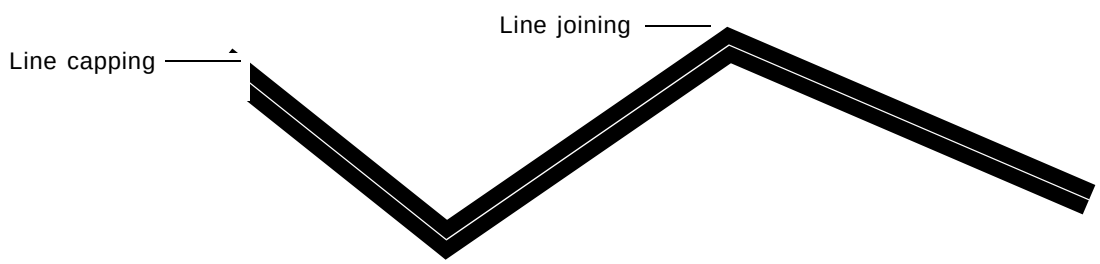
The following definitions describe the relationship between a polygon's path and its border.

Border centering: In IPL, the width of an object's border is centered on its path. For example, if a border is four points wide, two points of the border are on one side of the path, and two points are on the other. Border centering applies to all objects (including curved objects).

Border capping: The current line cap style determines the style to use for the ends of an object's borders. For a complete description of line cap styles, see "linecap" on page 13.

Border joining: A Frame product joins border segments so the angle of the border matches the angle of the path (in PostScript, this is called *miter join*). For example, if two segments of a polyline's path meet at a 30 degree angle, the angle of the polyline's border is also 30 degrees.

The following illustration shows a greatly expanded polyline with four points. Notice that the polyline is centered on its path, its ends have butt caps, and the line segments are joined:



Arc

`Arc` *`PosX PosY Width Height StartAngle DrawAngle`*

`PosX`	Real
`PosY`	Real
`Width`	Real
`Height`	Real
`StartAngle`	Real
`DrawAngle`	Real

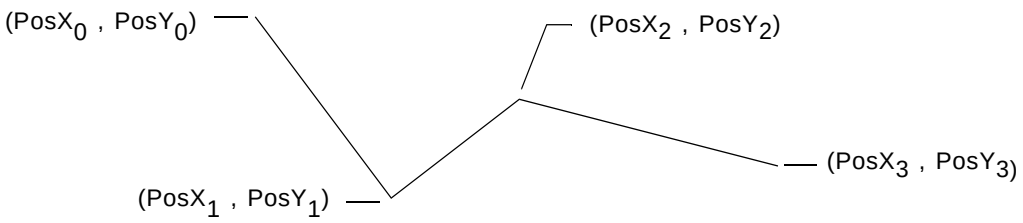
The `Arc` command describes an arc drawn with the current pen pattern and filled with the current fill pattern. *`PosX`*, *`PosY`*, *`Width`*, and *`Height`* define a rectangle containing an ellipse on which the arc lies. *`StartAngle`* is the angle (in degrees) from a vertical line bisecting the ellipse to the point where the arc begins. *`DrawAngle`* is the angle from the start point of the arc to the end point (or how many degrees to sweep through to create the arc). Positive angles extend clockwise around the center of the ellipse; negative angles extend counterclockwise.

Polyline

`Polyline` *`NumberOfPoints [PosX`*$_1$ *`PosY`*$_1$ *`PosX`*$_2$ *`PosY`*$_2$ *`…PosX`*$_n$ *`PosY`*$_n$*`}`*

`NumberOfPoints`	Integer
`PosX`	Real
`PosY`	Real

The `Polyline` command defines a set of connected line segments. The *`NumberOfPoints`* parameter is the number of points in the polyline. A set of *`PosX`* and *`PosY`* parameters defines the coordinates of each point. The polyline is drawn with the current pen pattern (set with the `Pen` command). Here's an example of a simple polyline with four points:

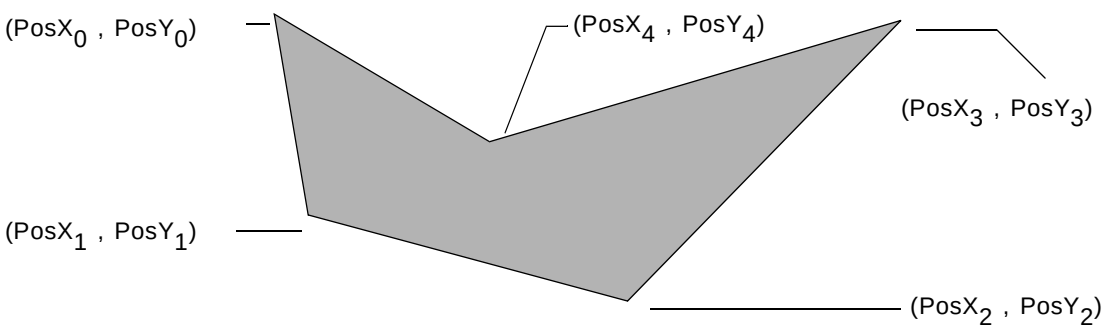


Polygon

Polygon *NumberOfPoints [PosX PosY]*$_{NumberOfPoints}$

NumberOfPoints	Integer
PosX	Real
PosY	Real

The Polygon command is similar to the Polyline command. It defines a set of connected line segments forming a closed shape that can contain a fill pattern. The shape is closed because the final line segment is implicitly drawn from the last coordinate to the first coordinate. The *NumberOfPoints* parameter is the number of points in the polygon. A pair of *PosX* and *PosY* parameters defines the coordinates of each point. The polygon is first filled with the current fill pattern (set with the Fill command) and then outlined with the current pen pattern (set with the Pen command). Here's an example of a simple polygon with five points:



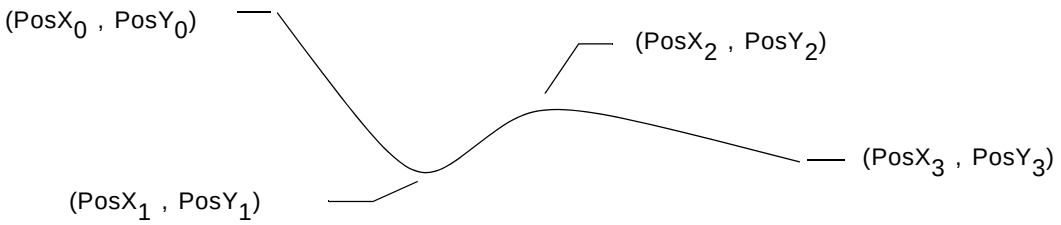
Smoothline

Smoothline *NumberOfPoints [PosX PosY]*$_{NumberOfPoints}$

NumberOfPoints	Integer
PosX	Real
PosY	Real

The Smoothline command is similar to the Polyline command, except that it defines a smoothed polyline. The *NumberOfPoints* parameter is the number of points in the line; it is always equal to 3*n*+1, where *n* is the number of Bezier segments in the line. A pair of *PosX* and *PosY* parameters defines the coordinates of each point your driver should use in the Bezier calculation to perform the smoothing. The line is drawn with the current

pen pattern (set with the Pen command). Here's an example of a simple smoothed line with four points (the labels point to the position of the points before smoothing):



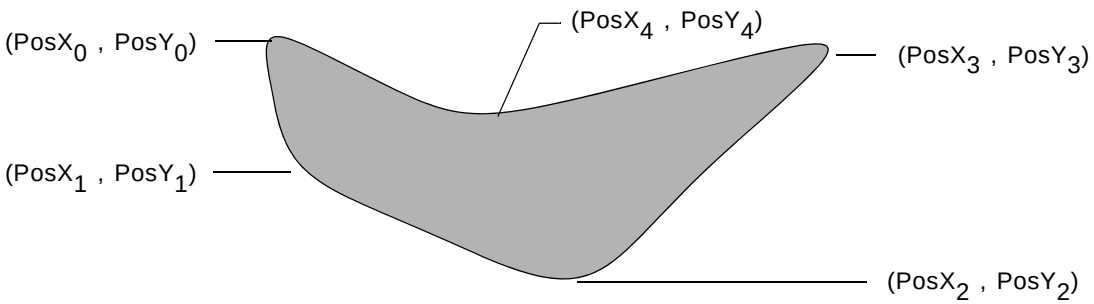
Smoothgon

Smoothgon *NumberOfPoints [PosX PosY]*$_{NumberOfPoints}$

NumberOfPoints	Integer
PosX	Real
PosY	Real

The Smoothgon command is similar to the Polygon command, except it defines a smoothed polygon. The *NumberOfPoints* parameter is the number of points in the polygon; it is always equal to 3*n*+1, where *n* is the number of Bezier segments in the line. A pair of *PosX* and *PosY* parameters defines the coordinates of each point your driver should use in the Bezier calculation to perform the smoothing. The polygon is first filled with the current fill pattern and then outlined with the current pen pattern.

Here's an example of a simple smoothed polygon with five points (the labels point to the position of the points before smoothing):

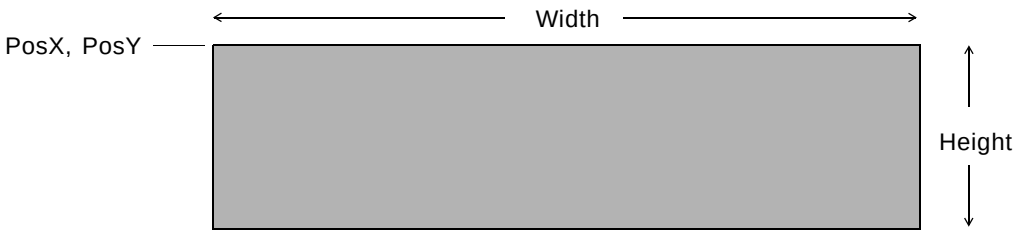


Rectangle

Rectangle *PosX PosY Width Height*

PosX	Real
PosY	Real
Width	Real
Height	Real

The Rectangle command defines a rectangle that is both filled (with the current fill pattern) and outlined (with the current pen pattern). *PosX* and *PosY* are the coordinates of the upper-left corner of the rectangle. *Width* and *Height* are the dimensions (in printer's points) of the rectangle.



The Rectangle command defines a rectangle that is both filled and outlined.

As with all outlined objects, when the width of its border increases, the object becomes bigger because the border is centered on the path describing the border.

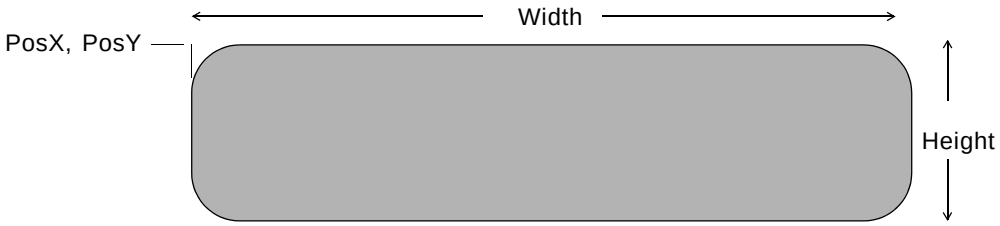
The Rectangle command is one of four commands that define rectangles. (The others—RoundRect, PenRectangle, and FillRectangle—are described next.) For all rectangles, you can avoid rounding errors in line placement and thickness by normalizing the values of *PosX*, *PosY*, *Width*, and *Height* before calculating the other corners of the rectangle.

RoundRect

RoundRect *PosX PosY Width Height Radius*

PosX	Real
PosY	Real
Width	Real
Height	Real
Radius	Real

The `RoundRect` command defines a filled and outlined rectangle with rounded corners. It has the same syntax as the `Rectangle` command with one additional parameter: *`Radius`*. *`Radius`* is the radius (in printer's points) of the rectangle's corners.



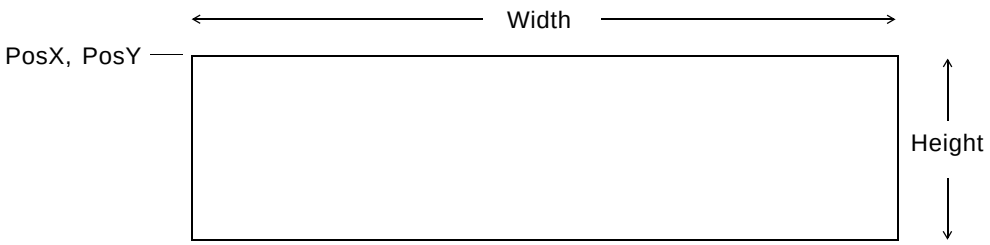
The RoundRect command defines a rectangle with rounded corners.

PenRectangle

`PenRectangle` *`PosX PosY Width Height`*

`PosX`	Real
`PosY`	Real
`Width`	Real
`Height`	Real

The `PenRectangle` command has the same syntax as the `Rectangle` command. PenRectangle defines a rectangle that is outlined, but not filled. (`Rectangle`, on the other hand, both fills and outlines the rectangle.)



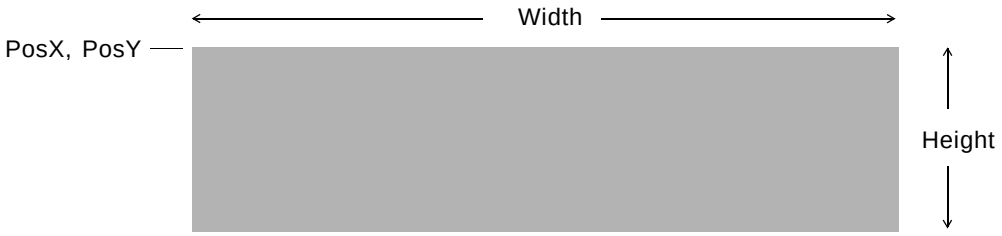
The PenRectangle command defines a rectangle that is outlined, but not filled.

FillRectangle

`FillRectangle` *`PosX PosY Width Height`*

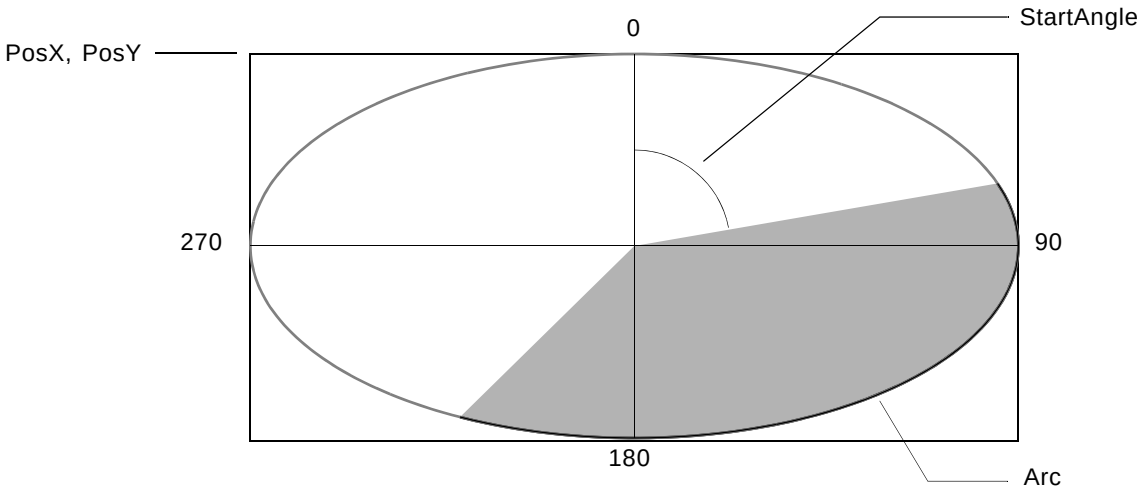
`PosX`	Real
`PosY`	Real
`Width`	Real
`Height`	Real

`FillRectangle` has the same syntax as the `Rectangle` command. `FillRectangle` defines a rectangle that is filled, but not outlined.



The FillRectangle command defines a rectangle that is filled, but not outlined.

For example, consider this arc, defined with a *StartAngle* of 71 degrees and a *DrawAngle* of 136 degrees (the arc to be drawn is filled):



Imported graphics commands

IPL has several commands for representing imported graphics and insets.

Bitmap

`Bitmap` *`Invert DotsPerInch PosX PosY Width Height FileName`*

`Invert`	Always 0
`DotsPerInch`	Integer
`PosX`	Real
`PosY`	Real
`Width`	Real

Height	Real
FileName	Qstring

The Bitmap command defines a graphics file or external inset to be printed. Your printer driver should read the file header to determine its type. It might also need to convert the image to a format supported by your printer driver.

The *Invert* parameter is always 0. The *DotsPerInch* parameter determines whether the image is to be scaled using a set scaling factor or scaled to fit in a rectangle with the dimensions of *Width* and *Height*. If *DotsPerInch* is not 0, the value of *DotsPerInch* is the image's scaling factor (its printing resolution). In this case, scale the image so each dot in the image takes up 1/*DotsPerInch* inches. Typically, this value is an integral divisor of the printer's resolution.

If *DotsPerInch* is 0, scale the image so it fits in the rectangle defined by *Width* and *Height*. *PosX*, *PosY*, *Width*, and *Height* define a rectangle containing the image, setting its position on the page.

FileName is the name of the file holding the graphics file. The *FileName* parameter gives the UNIX filename for the graphics file.

InlineBitmap

InlineBitmap *Invert DotsPerInch PosX PosY Width Height*

Invert	Always 0
DotsPerInch	Always 0
PosX	Real
PosY	Real
Width	Real
Height	Real

The InlineBitMap command defines an internal inset or imported graphic that was copied into the Frame document. Its parameters are the same as the BitMap command. *Invert* and *DotsPerInch* are always 0.

A FrameImage facet describing the image follows the *rotate Angle* parameter. It begins with the lines:

```
=FrameImage
&%v
```

Following these two lines is a complete description of the image in FrameImage format. You may need to translate the image to a format supported by your printer.

The following line marks the end of the facet (the end of the InlineBitmap command):

```
=EndInset
```

BinPrintCode

BinPrintCode *PosX PosY Width Height MinusX MinusY MinusW MinusH*

PosX	Real
PosY	Real
Width	Real
Height	Real
MinusX	Real
MinusY	Real
MinusW	Real
MinusH	Real

The BinPrintCode command defines an EPSI (Encapsulated PostScript Interchange format) internal inset or imported EPSI graphic that was copied into the Frame document. The Frame product writes this command to an IPL file only when the current print driver is fmprintdriver.ps (when you are printing to a PostScript printer). Otherwise it writes an InlineBitmap command describing the image. (See "InlineBitmap" on page 27.)

PosX, *PosY*, *Width*, and *Height* define a rectangle that your driver can use to position the image on the page, but doesn't have to (some printer code defines its own position within the code). The final four parameters can be safely ignored by your printer driver.

An EPSI facet describing the image follows the *MinusH* parameter. It begins with the lines:

```
=EPSI
&%v
```

Following these two lines is a complete description of the image in EPSI format. The following line marks the end of the facet (the end of the BinPrintCode command):

```
=EndInset
```

Miscellaneous commands

In addition to text and graphics descriptions, an IPL file can contain comments and printer code in your page description language.

% (comment)

The % command indicates a comment. Your printer driver should ignore all lines that begin with a %. Also, an IPL file occasionally contains a blank line. Your driver should skip all blank lines.

PrintCode

PrintCode *NumberOfLines PosX PosY Width Height MinusX MinusY MinusW MinusH*

NumberOfLines	Integer
PosX	Real
PosY	Real
Width	Real
Height	Real
MinusX	Real
MinusY	Real
MinusW	Real
MinusH	Real

A Frame product document can contain a series of PostScript commands. The IPL PrintCode command passes PostScript commands to the printer driver unchanged. Your printer driver can handle this command in any way you want. For example, you might ignore the PostScript code, translate the PostScript code into your page description language, or instruct users to use your page description language in place of PostScript commands.

The *NumberOfLines* parameter sets the number of lines immediately following that are in printer code. *PosX*, *PosY*, *Width*, and *Height* define a rectangle that your driver can use to position the image on the page. Some printer code defines its own position within the code. The final four parameters can be safely ignored by your printer driver. They contain the negative values of the previous four parameters. These parameters were added to provide backward compatibility and might not be in future versions of IPL.

An include statement, within the lines of printer code, references an external printer code file. It has the format:

#include "*filename*"

where *filename* is the name of the external printer code file. The quotation marks should appear in the IPL file.

3 Setting Up to Call Your Driver3

The `fminit` directory contains files that set up standard Frame product features and printer-specific information such as the default printer driver and available fonts. Because most of the information is printer-specific, `fminit` is actually a symbolic link to a printer-specific `fminit` directory. For example, when a Frame product is set up to use the PostScript language, `fminit` is a symbolic link to the directory `fminit.ps`. This chapter tells you how to set up an fminit directory for your page description language. It also describes the parameters a Frame product passes to your printer driver.

The remainder of this manual uses the term `$FMHOME` to refer to the directory in which a Frame product is installed. Whenever you see this term, substitute the pathname of the Frame product installation directory for your system.

Creating an fminit directory

To create an fminit directory for your printer driver, copy `fminit.ps` (the standard Frame product setup directory) and remove some of the files from it.

1. **Copy $FMHOME/fminit to $FMHOME/fminit.pdl, where pdl is a standard suffix for your page description language.**
2. **Remove the following PostScript-specific files from fminit.*pdl:***

   `fmfontconfig`

   `ps_prolog`

   `FMlpr (also FMlpr.fax, FMlpr.finder, and so on)`
3. **Remove all of the files from the directory fminit.*pdl*/fontdir.**

   For instructions on adding fonts for your page description language to this directory, see Chapter 4, “Adding Fonts.”

Setting up to use your fminit directory

To set up a Frame product to use your `fminit` directory, set up a symbolic link from `fminit` to your `fminit.pdl` directory. Use the following commands:

```
cd $FMHOME
rm fminit
ln -s fminit.pdl fminit
```

Frame recommends you put the above commands in a configuration shell script so users can easily switch from one page description language to another. Frame product users can run this script to configure a Frame product to use your page description language. The

following script creates symbolic links to files for an imaginary page description language called PDL:

```
#!/bin/sh
# Reconfigures fminit for PDL by resetting symbolic
# links to fminit

cd $FMHOME
rm fminit
ln -s fminit.pdl fminit
```

The `fminit.ps` directory contains a similar shell script named `fmPostScriptConfigure`. Following similar naming conventions, we would call the above script `fmPDLConfigure`.

You may want to copy your configuration script to the `fminit.ps` directory so that users of the PostScript-language version of a Frame product can easily switch to your page description language.

Setting up to call your printer driver

The printer file in `fminit` controls which printer driver a Frame product uses, the preset printer name, and the preset paper dimensions. The standard printer file contains the following information:

```
Default printer to use:
  <Printer ps > (NB trailing space necessary)
Default printer paper size to use:
  <PaperSizeWidth 8.5" >
  <PaperSizeHeight 11.0" >
Printer PDL extension:
  <PrinterLanguage ps >
```

Edit the printer file in `fminit.pdl` so that it calls your printer driver and contains appropriate settings for your printer. The <Printer> statement contains the preset printer name that a Frame product displays in the Print dialog box. Edit this statement so that it contains the standard printer name for your page description language.

The <PaperSizeWidth> and <PaperSizeHeight> statements contain the preset paper dimensions. If necessary, edit these statements so that they contain the standard paper dimensions for your page description language.

A Frame product uses the <PrinterLanguage> statement to determine which printer driver to call. Edit this statement so that it contains the standard suffix for your page description language. Use the same abbreviation that you used to name your `fminit` directory.

Storing your printer driver

When a user prints a document, the Frame product creates an IPL file describing the document. It then looks up the page description language suffix in `fminit/suffix` and starts the program `$FMHOME/bin/fmprintdr.pdl,` where it translates the IPL file into a page description language for a specific printer.

Copy your printer driver to the directory `$FMHOME/bin`. Give it the name `fmprint.pdl`, where `.pdl` is the standard suffix for your page description language.

Printer driver parameters

A Frame product executes printer drivers with the following arguments in this order:

Parameter:	Description:
`IPLFileName`	The full name of the IPL file
`PrinterName`	The name of the printer where the results are to be sent (from the Printer Name box in the Print dialog box)
`SendToPrinter`	A zero/one flag indicating whether the results should be sent to the printer, or copied back on top of the IPL file
`TemporaryFileName`	A temporary filename a Frame product creates that won't conflict with any other Frame product print job and is thus suitable for temporary storage
`UserHomeDirectoryPath`	The path to the user's home directory
`fminitPath`	The path to the `fminit` directory
`binDirectoryPath`	The path to the Frame product `bin` directory (`$FMHOME/bin`)

Any messages that the printer driver sends to its `stdout` appear in the window from which you started `$FMHOME/bin/fmprintdr.pdl`.

4 Adding Fonts

This chapter describes how to add fonts for your page description language so a Frame product is able to make line-break decisions and display fonts accurately on the screen. As in the preceding chapter, it also describes the changes you can make to files in `fminit` so that a Frame product uses the correct font files for your page description language.

If you are adding PostScript fonts to a Frame product, do not follow the instructions in this chapter. Instead, see the online manual, *Managing Frame Fonts*.

Creating font files

To customize a Frame product for use with a different font set, you need information about character widths so a Frame product can make reasonable line-breaking decisions, and font glyphs so a Frame product can display its results on the screen. The width information is held in AFM (Adobe Font Metric) files, and the font glyphs are in bfont files. Both font file formats are described next.

The AFM files

AFM files contain typographic information a Frame product uses to decide how to break lines and how to kern characters. For a complete description of AFM format, see *Adobe Font Metric Files Specification,* available from Adobe Systems Inc.

The bfont files

A bfont file holds an image of the entire set of characters in a font at a specific point size and for a specific screen resolution or scale factor. The file includes a header, followed by a single glyph of all the character images, side-by-side. The `bfont` file format is:

Type:	Name:	Definition:
U2	size	Character size, in points
U2	filler	Ignored
U2	base	Scan lines from top of tallest character to baseline
U2	height	Scan lines from top of tallest to depth of deepest character
U2	xheight	Scan lines from baseline to xheight
U2	ul_y	Scan lines from baseline to underline position
U2	ul_t	Scan lines of underline thickness
U2	rleft	The left origin of the font set glyph's bounding box

Type:	Name:	Definition:
U2	rtop	The bounding box height minus the bounding box origin
U2	rwidth	The width of the bounding box
U2	rheight	Scan-line height of glyph image
U2	rwords	Width of glyph, in 16-bit words
U4	filler	Ignored
X6	bcp[256]	Character pointer information
U2	image[rwords*rheight]	

The glyph represents the entire font, end-to-end, with the bytes of the top scan-line appearing first, left-to-right, high-order to low-order bits. Types abbreviated U*n* are unsigned integers of *n* bytes. The image includes no blank space for side-bearings because the bcp array makes this unnecessary. The X6 type indicates a four-field data structure that looks like:

Type:	Name:	Definition:
U2	x	Bit offset to upper-left bit of character image within image array
U2	w	Bit width of character image within image array
U1	dx	Character escapement (all inclusive origin-to-origin)
U1	kx	Left side-bearing (white columns not appearing in character image)

Characters without glyphs have x = -1 and w = 0 (for example, Tab and Return). Nonexistent characters have dx = 0 and kx = 0.

Font filenames

Your font files should follow naming conventions similar to those used by PostScript fonts. The official font name is the font face (for example, `Helvetica-Bold` and `Helvetica-BoldOblique`). AFM filenames should be *`Official_font_name`*`.afm` (for example, `Helvetica-Bold.afm`) and `bfont` filenames should be *`Official_font_namePoint_size`*`.bfont` (`Helvetica-Bold12.bfont`).

Installing TypeScaler fonts

TypeScaler® fonts are screen fonts that supply a closer match between text you see on the screen and printed text, especially at larger and nonstandard sizes. When TypeScaler fonts are installed, a Frame product uses them whenever they provide greater clarity than the standard screen fonts. TypeScaler fonts are available from Sun Microsystems.

To install TypeScaler font files that correspond to your fonts, copy them into the directory `$FMHOME/fminit.`*`pdl`*`/fontdir`. In the Frame product font directory, the TypeScaler font files should have names beginning with the official font family name, followed by a period

and the suffix `f3b`. For example, the TypeScaler font file used to display Courier fonts has the name `Courier.f3b`.

Configuring a Frame product to use font files

After you create the AFM and bfont files, you must tell a Frame product where to look for the font files. You configure a Frame product to use your font files by moving them to a special subdirectory in your `fminit` directory. Then you create a `fontlist` file describing each of the font files you created.

Moving your font files to the font directory

A Frame product looks for font files in the directory `$FMHOME/fminit/fontdir`. The `fminit` directory is a symbolic link to your printer-specific `fminit` directory: `fminit.pdl`. (See Chapter 3, "Setting Up to Call Your Driver.") Move your AFM and bfont files into `$FMHOME/fminit.pdl/fontdir`.

Creating a fontlist file

The `fontlist` file in `$FMHOME/fminit.pdl/fontdir` describes your fonts for a Frame product. It consists of a series of statements describing font sizes, families, variations, weights, angles, font files, and foreign font mappings. These statements determine your font property choices in the Paragraph Character Designer windows. You choose a font by specifying each of the font properties.

The `fontlist` describes available fonts in terms of these properties, so that when you specify Font properties in one of these windows, a Frame product knows which AFM and bfont files to use to display or print the fonts. To add fonts to the `fontlist`, you add statements describing them.

Because the `fontlist` file is a standard ASCII text file, you can edit it with any standard text editor. If you use a Frame product, be sure to save the document text as an ASCII text file (by choosing Text Only in the Save As dialog box).

Font size section

The font size section contains several <Size> statements and one <DefaultSize> statement. For example:

```
<Size 7 pt>
<Size 8 pt>
<Size 9 pt>
<Size 10 pt>
<Size 11 pt>
<Size 12 pt>
<Size 14 pt>
```

```
<Size 18 pt>
<Size 24 pt>
<Size 36 pt>
<DefaultSize 12 pt>
```

Point sizes listed in <Size> statements appear, in order, in the Size pop-up menu. The size listed in the <DefaultSize> statement is the default point size. The `fontlist` does not limit your font size choices in a Frame document; you can choose a font size from the pop-up menu, or you can type any size you want. A Frame product uses the appropriate AFM and bfont files to display and print the font. If there is no bfont file for a particular size, a Frame product uses a corresponding TypeScaler font (see "Installing TypeScaler fonts" on page 2) or it uses the font information for the next smaller size and then scales the font to the size you choose.

Family section

The family section contains several types of statements that describe font families. The family section of a PostScript `fontlist` usually looks something like this:

```
<Family AvantGarde >
<Family Bookman >
<Family Courier >
<Family Helvetica >
<Family LucidaBright >
<Family LucidaNewMath >
<Family NewCenturySchlbk >
<Family Palatino >
<Family Symbol >
<Family Times >
<Family ZapfChancery >
<Family ZapfDingbats >
<ForeignFamily Helvetica-Narrow >
<ForeignFamily Lucida >
<ForeignFamily LucidaSans >
<ForeignFamily Geneva >
<ForeignFamily NewYork >
<ForeignFamily Monaco >
<DefaultFamily Times >
<NonText Symbol >
<NonText ZapfDingbats >
```

```
<NonText LucidaNewMath >
<MathFamily Symbol >
<FrameFamily Frame >
```

Font families listed in <Family> statements appear (in the order in which they appear in the `fontlist`) as choices in the Font Family scroll list. Add a <Family> statement for each font family that you stored in the `fontdir` directory.

In a <DefaultFamily> statement, name the default font family. A Frame product uses the default family when no other family has been assigned to text.

<NonText> statements tell which families do not contain alphanumeric characters and therefore should be ignored by the spelling checker. For example, in the list above, the Symbol family is listed as a nontext family because it contains no regular alphanumeric characters. If one of the new fonts contains nontext characters, add a <NonText> statement for it to the `fontlist`.

In the <MathFamily> statement, name the font family to use for math.

> **Important:** The <FrameFamily> statement describes a special family a Frame product uses for text symbols. This statement must read:
>
> <FrameFamily Frame>

Variation section

The variation section contains a series of <Variation> statements. For example:

```
<Variation UltraCompressed >
<Variation ExtraCompressed >
<Variation Compressed >
<Variation Condensed >
<Variation Narrow >
<Variation Regular >
<Variation Wide >
<Variation Poster >
<Variation Expanded >
<Variation Arrows >
<Variation Extension >
<Variation Symbol >
<DefaultVariation Regular >
```

Each <Variation> statement describes a choice in the Variation pop-up menu, and the <DefaultVariation> statement names the default variation. Choices appear in the menu in the same order that they appear in the `fontlist`. Add a statement for each variation provided by your fonts.

Weight section

The weight section contains a series of <Weight> statements. For example:

```
<Weight UltraLight >
<Weight ExtraLight >
<Weight Thin >
<Weight Light >
<Weight Regular > <WeightAlias Roman Regular > <WeightAlias Medium
Regular >
<Weight Book >
<Weight SemiBold > <WeightAlias Semi SemiBold >
<Weight DemiBold > <WeightAlias Demi DemiBold >
<Weight Bold > <WeightAlias Bolded Bold >
<Weight ExtraBold >
<Weight Heavy >
<Weight Black >
<DefaultWeight Regular >
```

Each <Weight> statement describes a choice in the Weight pop-up menu, and the <DefaultWeight> statement names the default weight. Choices appear in the menu in the same order that they appear in the `fontlist`. Add a statement for each weight provided by your fonts. Use a light-to-bold order.

Angle section

The angle section contains a series of <Angle> statements. For example:

```
<Angle Regular >
<Angle Kursiv >
<Angle Slanted >
<Angle Oblique > <AngleAlias Obliqued Oblique >
<Angle Italic >
<DefaultAngle Regular >
```

Each <Angle> statement describes a choice in the Angle pop-up menu, and the <DefaultAngle> statement names the default angle. Choices appear in the menu in the same order that they appear in the `fontlist`. Add a statement for each angle provided by your fonts.

Font section

The font section contains <Font> statements that list official font names and the family, variation, weight, and angle that correspond to them. For example:

```
<Font     F-R      Frame >
<Font     F-I      Frame Italic >
<Font     F-B      Frame Bold >
<Font     F-BI     Frame Bold Italic >

<Font     Cr       Courier >
<Font     Cr-O     Courier Oblique >
<Font     Cr-B     Courier Bold >
<Font     Cr-BO    Courier Bold Oblique >

<Font     Tim-R    Times >
<Font     Tim-I    Times Italic >
<Font     Tim-B    Times Bold >
<Font     Tim-BI   Times Bold Italic >

<Font     H        Helvetica >
<Font     H-O      Helvetica Oblique >
<Font     H-B      Helvetica Bold >
<Font     H-BO     Helvetica Bold Oblique >
<Font     H-N      Helvetica Narrow >
<Font     H-N-O    Helvetica Narrow Oblique >
<Font     H-N-B    Helvetica Narrow Bold >
<Font     H-N-BO   Helvetica Narrow Bold Oblique >

<Font     Symbol   Symbol >
. . .
```

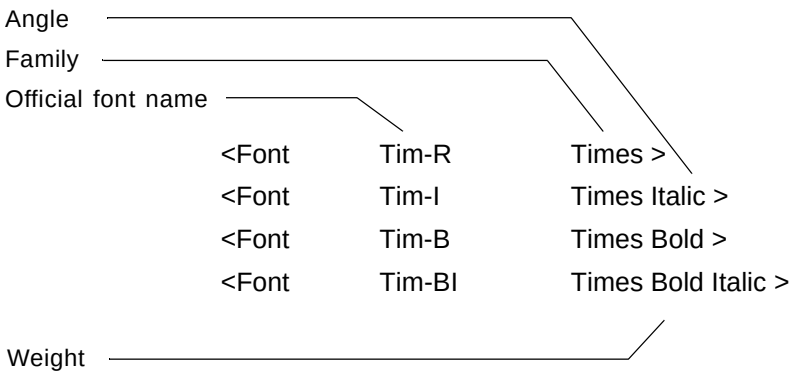
A Frame product uses the <Font> statements to look up the font files for the font properties you choose when editing a document. It also uses <Font> statements to limit your variation, weight, and angle choices to ones that are available for the font family you choose.

The `fontlist` must contain a <Font> statement for each official font name. The <Font> statement contains the font name and the font properties that describe the font. If you do not list a font property value in the <Font> statement, the font uses the default value for that property. For example, the following line tells a Frame product to use the font Times-Italic

when the font family is Times, the angle is Italic, and the weight and variation properties have their default values (Regular):

<Font Tim-I Times Italic >

Add a <Font> statement for each font. For example, you would use the following <Font> statements to describe the PostScript Garamond fonts:



Status section

The status section contains a series of <Status> statements. There are two status statements: <StatusLight> and <StatusHeavy>. <StatusLight> names the face for rulers; <StatusHeavy> is for Frame internal use only. Frame recommends that you use a sans serif face, such as the Helvetica face, for <Status> statements.

Foreign font section

Users who print to one printer might not use the same `fontlist` as users who print to another printer. For example, the printer in your documentation department might use the standard PostScript fonts, while the printer in your engineering department uses fonts for another page description language; the writers and the engineers use different `fontlist` files. When users who do not use the same `fontlist` share documents, those documents can specify unavailable fonts. When a Frame product encounters unavailable (foreign) fonts, it uses the closest available font. You can give a Frame product specific instructions for handling unavailable fonts by editing the `fontlist` file.

The foreign font section of the `fontlist` contains two types of statements that tell a Frame product how to substitute fonts. <ForeignFamily> statements name font families that are not available in this `fontlist`, but might be specified in documents that were created using another `fontlist`. See page 4. <ForeignFamily> statements have the following format:

`<ForeignFamily` *`ForeignFontFamilyName`*`>`

<MapFont> statements tell which available font to substitute for a foreign font. They use Font properties to describe font faces for both the foreign and the native font. Each <MapFont> statement has the following format:

```
<MapFont <ForeignFontPropertyList> <NativeFontPropertyList> >
```

If all of the users at your site use the same `fontlist`, you do not need to add any <ForeignFamily> and <MapFont> statements when you add fonts.

If some users will use a different `fontlist`, you should add <ForeignFamily> and <MapFont> statements describing the new fonts to their `fontlist`. For example, when some users will continue to use PostScript fonts and others will use the fonts for the new page description language, add the new fonts to the PostScript `fontlist`. This way, when a user of the PostScript `fontlist` opens a document that specifies a new font (for example, HelvPDL), a Frame product can substitute an available font (for example, Helvetica). To map the HelvPDL faces to Helvetica faces, for example, add the following statements to the map font section of the PostScript `fontlist`:

```
<ForeignFamily HelvPDL>
<MapFont <HelvPDL Light> <Helvetica> >
<MapFont <HelvPDL Bold> <Helvetica Bold> >
<MapFont <HelvPDL Light Italic> <Helvetica Italic> >
<MapFont <HelvPDL Bold Italic> <Helvetica Bold Italic> >
```

If users of your new `fontlist` need to open documents created using a different `fontlist`, add <ForeignFamily> and <MapFont> statements to the new `fontlist`. For example, if users of the new `fontlist` need to open PostScript documents, add a <ForeignFamily> statement for each standard PostScript font family and then add <MapFont> statements that substitute the new fonts for the PostScript fonts.

For example, to substitute HelvPDL fonts for PostScript Helvetica fonts, add the following statements to the foreign font section:

```
<ForeignFamily Helvetica>
<MapFont <Helvetica> <HelvPDL Light> >
<MapFont <Helvetica Bold> <HelvPDL Bold> >
<MapFont <Helvetica Italic> <HelvPDL Light Italic> >
<MapFont <Helvetica Bold Italic> <HelvPDL Bold Italic> >
```

See the PostScript `fontlist` file in `$FMHOME/fminit.ps/fontdir` for a list of standard PostScript fonts.

Folio section

A folio section may follow the foreign font section. The <Folio> statements in this section list TypeScaler fonts that a Frame product can use to display the standard fonts for your page description language on the screen. (See “Installing TypeScaler fonts” on page 2.) When TypeScaler fonts are installed, a Frame product uses them whenever they provide greater clarity than the standard screen fonts. Add a <Folio> statement for each TypeScaler

font you installed. Each <Folio> statement lists the Font Family and the TypeScaler font file. For example, the <Folio> statement for the Courier TypeScaler font is:

<Folio Cr Cr.13b >
<Folio Cr-O Cr-O.f3b >
<Folio Cr-B Cr-B.f3b >
<Folio Cr-BO Cr-BO.f3b >

Checking your work

After creating the `fontlist` file, you're ready to see if you can use the new fonts.

1. **Restart your Frame product.**
2. **Open a new document.**
3. **Display the Character Designer window.**

   The font families you added should appear in the scroll list.
4. **Type some text and verify that you can change it to the various sizes and styles you put into the fontlist.**

PostScript and platform names of fonts

When a Frame product reads a MIF file that includes more than one way of identifying a font, it checks the font name in the following order:

1. Platform name
2. Combination of family, angle, weight, and variation properties
3. PostScript name

If you are writing filters to generate MIF, you do not need to use all three methods. You should always specify the PostScript name, if it is available. You should use the platform name only if your filter will be run on a specific platform. A filter running on a specific platform can easily find and write out the platform name, but the name cannot be used on other platforms.

For information on PostScript and platform names, see *MIF Reference*.

Setting up templates

The standard templates and other on-line documents distributed with a Frame product use PostScript fonts. When you open one of these documents, your Frame product converts the PostScript fonts to the fonts for your page description language. Before you distribute your printer driver and `fminit.pdl` directory to Frame product users, you should open each on-line document and save it with the new fonts. Depending on how closely the fonts map, you might want to make additional changes to a document before you save it.

Check the following directories in `$FMHOME/fminit/`*`UILanguage`*, where *`UILanguage`* is one of the Frame product user interface languages:

- `CustomDocs`

  This directory contains the template for a new custom document (`NewTemplate`)

- `Samples`

  The standard template files (in the directory `Samples/Templates`) and a demonstration document (in the directory `Samples/DemoDoc`)

- `Maker`

  The help files (in the directory `Maker/Help`) for the specific language.

Go To

IPL Reference Index

R

S

T