KL Group PageFormatter™

Programmer's Guide & Reference Manual

Version 2.0



Software Development Productivity

260 King Street East
Toronto, Ontario, Canada M5A 1K3
(416) 594-1026
www.klg.com

September 1998

RefNo: PRGDE-KLPF/U/200-09/98

Copyright © 1997, 1998 by KL Group Inc. All Rights Reserved.

KL Group, the KL Group logo, and PageFormatter are trademarks or registered trademarks of KL Group Inc. in Canada, United States and other countries.

All other products, names, and services are trademarks or registered trademarks of their respective companies or organizations.

Printed in Canada on recycled paper. 

A Note on KLGPR Product Licensing

The terms and conditions of the KLGPR product license are detailed in the license agreement on the next page. Please take a few minutes to read and understand the agreement. The notes below are offered to aid in reading the agreement only, and are not considered part of the agreement itself.

- KLGPR products (such as KL Group PageFormatter) are licensed on a per architecture (such as SPARC, RS6000, HP 700/800, etc.) basis.
- There are two types of licenses: single and network-site.
- A single license authorizes use of a KLGPR product for development of end-user applications on a particular machine identified by a CPU ID. Development activities include compiling and linking the KLGPR product library into an end-user application binary.
- A network-site development license authorizes use of a KLGPR product for development of end-user applications on all similar-architecture machines on the same network and located in the same building.
- You may distribute end-user application binaries including the statically-linked KLGPR product library anywhere, and as many times as you like, without incurring royalty or run-time fees.

KLGPR BINARY LICENSE AGREEMENT

Important — Read Carefully Before Opening Software Package(s), Installing or Using Software On Your Computer

By opening a sealed software package, or installing, or using this KLGPR product, you indicate your acceptance of the following KLGPR Binary License Agreement. This is a legal agreement between you (either an individual or an entity) and KL Group Inc. If you do not accept and agree with all of the terms of this agreement, promptly destroy any downloaded or installed files, and return the unopened software package(s) and all other materials with proof of payment to your place of purchase, and your license fee will be refunded.

SOFTWARE LICENSE

Please Note: This is a license agreement and **not** an agreement for sale.

Grant of License. This License Agreement ("License") permits you to use one copy of the software you purchased (the "SOFTWARE"), on a single computer after registration of that computer's CPU identification with us. If you have purchased a network site license you may also use the SOFTWARE on computers in the same building having the same architecture as a registered computer, and connected by the same local area network, provided that the total number of such computers, on which the SOFTWARE is ever used, does not exceed the maximum number permitted by the network site license which you ordered. The SOFTWARE is "in use" on a computer when it is loaded into temporary memory (i.e. RAM); when it is installed into permanent memory (e.g. hard disk, CD-ROM, or other storage device); when it is being linked into an end-user application; or when it is accessed from a software development tool such as a compiler, a linker, or a GUI builder. The SOFTWARE is not considered "in use" when installed on a network file server for the sole purpose of file distribution to other computers.

Updates. This agreement will govern the terms of any SOFTWARE updates provided to you by KL Group.

Copyright. The SOFTWARE (including any images, photographs and text incorporated into the SOFTWARE) is owned by KL Group or its suppliers and is protected by Canadian and United States copyright laws and international treaty provisions. Therefore you must treat the SOFTWARE like any other copyrighted material and not reproduce it except that you may make a backup for archival purposes. You may not copy the printed materials accompanying the SOFTWARE.

Other Restrictions. You may not rent, lease, transfer, reverse engineer, decompile, disassemble, create passwords for or translate the Software, except to the extent such foregoing restriction is expressly prohibited by applicable law.

Distribution of applications you build with the SOFTWARE. You have a royalty-free right to reproduce and distribute files in the SOFTWARE that are found in the "lib" directory and which are in a form suitable for linking into object code application files (the "KLGPR Library Files") provided that (a) you distribute the KLGPR Library Files only when linked into your software product in such a way as to prohibit further relinking; (b) your software product is not an application development tool that provides similar functionality to the SOFTWARE licensed hereunder; (c) you do not use KL Group's name, logo or trademark to market your software product; (d) you include a valid copyright notice on your software product; and (e) you agree to indemnify, hold harmless, and defend KL Group and its suppliers from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of your software product.

U.S. Government Restricted Rights. The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software — Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is KL Group Inc., 260 King Street East, Toronto, Ontario, Canada, M5A 1K3.

LIMITED WARRANTY

Limited Warranty. KL Group warrants that the SOFTWARE will perform substantially in accordance with the accompanying written materials for a period of sixty (60) days from the date of receipt. Some states/jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. This warranty may not be assigned. KL Group does not exclude product liability or liability for damages resulting out of intent or gross negligence on the part of KL Group or its leading personnel.

Customer Remedies. KL Group's and its suppliers' entire liability and your exclusive remedy shall be, at KL Group's option, either (a) return of the price paid, or (b) repair or replacement of SOFTWARE that does not meet KL Group's Limited Warranty and which is returned to KL Group with a copy of your receipt. This Limited Warranty is void if failure of the SOFTWARE has resulted from accident, abuse, or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

Customer Remedies for Infringement. Should the SOFTWARE become or in KL Group's opinion be likely to become the subject of a claim for infringement of patent, trade secret or copyright, KL Group may (a) procure for you the right to continue to use the SOFTWARE; or (b) replace or modify the SOFTWARE to make it non-infringing, provided that the same function is performed by the replacement or modified SOFTWARE; or if (a) or (b) are not commercially reasonable, (c) terminate the license to use the SOFTWARE, remove the SOFTWARE, and grant you a credit for the license fee as depreciated on a straight-line five (5) year basis.

No Other Warranties or Obligations. To the maximum extent permitted by applicable law, KL Group and its suppliers shall have no other obligations and disclaim all other warranties, either express or implied, including, but not limited to, implied warranties of merchantability and fitness for a particular purpose, with regard to the SOFTWARE and the accompanying printed materials. This limited warranty gives you specific legal rights. You may have others which vary from state/jurisdiction to state/jurisdiction.

No Liability for Consequential Damages. To the maximum extent permitted by applicable laws, in no event shall KL Group or its suppliers be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of information, or other pecuniary loss) arising out of the use of or inability to use this KL Group product, even if KL Group has been advised of the possibility of such damages. Because some states/jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

Governing Law. If you are a resident of one of the member states of the European Union, this Agreement is governed by the laws of your country of residence, and each of the parties hereto irrevocably attorns to the jurisdiction of the courts of your country of residence. Otherwise, this Agreement is governed by the laws of the Province of Ontario, Canada. Each of the parties hereto irrevocably attorns to the jurisdiction of the courts of the Province of Ontario.

Public Domain Software. This license agreement and the warranties, remedies, limitations, conditions and restrictions contained herein do not apply to any software clearly marked as being in the public domain.

Should you have questions concerning this Agreement, or if you desire to contact KL Group for any reason, please write:

KL Group Inc., 260 King Street East, Toronto, Ontario, Canada, M5A 1K3
Phone (416) 594-1026 / Fax (416) 594-1919

Table of Contents

Preface 1
Introduction 1
Assumptions 1
Typographical Conventions Used in this Manual 1
Overview of Manual 2
What's New 2
Technical Support 3
Request for Comments 4
PageFormatter Announcements 4

Part I: Using KL PageFormatter

1 Developing a Simple KL Group PageFormatter Program 7
1.1 Setting Your Environment 7
1.2 Compiling and Authorizing a Program 8
1.3 Basic Flowing And Printing Techniques 8
1.4 Looking Ahead 12
1.5 Proceeding From Here 15

2 PageFormatter Basics 17
2.1 Basic Steps for Creating a PageFormatter Document 17
2.2 Objects 18
2.3 Attributes and Methods 21
2.4 Rendering Text with Flow and Frame Methods 22
2.5 Printing 24
2.6 Error Handling 24
2.7 Method Return Values On Errors 26

3 Document Object 27
3.1 Creating and Configuring the Document Object 27
3.2 Document Object Attributes 27
3.3 Controlling the Flow 30

3.4 Callbacks 32

4 Page Object 37

4.1 Page Object Attributes 37
4.2 Creating a Template Page Object 38
4.3 Defining a Template Page Object 38
4.4 Template Keywords 39
4.5 Parsing the Definition 43
4.6 Creation with KlgPfPageCreate() 43
4.7 Getting and Setting Page Attributes 44
4.8 Multiple Page Templates 44
4.9 Destroying a Page Object 47

5 Frame Object 49

5.1 Frame Object Attributes 49
5.2 Creating a Frame Object 50
5.3 Setting Frame Dimensions 50
5.4 Getting and Setting Other Frame Attributes 51
5.5 Frame Borders 51
5.6 Columns 53
5.7 Multiple Frames on a Page 54
5.8 Header and Footer Frames 54
5.9 Page Number Macros 59
5.10 Drawing Tools 60

6 Style Object 67

6.1 Style Object Attributes 67
6.2 Creating a Style Object 68
6.3 Copying a Style 68
6.4 Setting Style Attributes 68
6.5 Applying New Styles During Rendering 69
6.6 Changing Font Appearance 71
6.7 Changing Paragraph Appearance 75

7 Advanced Formatting 79

7.1 Tab Object 79
7.2 Color Object 82
7.3 Images 84

8 Tables 89

8.1 Table Object 89

8.2 Adding Data to the Table 93
8.3 Adding a Header 95
8.4 Changing the Table Appearance 96
8.5 Borders 100
8.6 Table Inheritance 104
8.7 Setting Row, Column, and Cell Attributes 104
8.8 Creating a Cell Span 107

9 Printer Object 109
9.1 Printer Object Attributes 109
9.2 Creating a Printer Object 110
9.3 PostScript Printer 112
9.4 PCL Printer 112
9.5 Printing Adobe Acrobat PDF Files 113
9.6 Printing to a Stream 113
9.7 Printing to a File 114
9.8 Output Policy 114
9.9 Callbacks 114
9.10 Print Previewer 115
9.11 Print Dialog Object 117

10 Lists 119
10.1 Creating and Destroying a List 119
10.2 Adding and Deleting List Items 120
10.3 Searching and Retrieving List Items 121
10.4 Sorting a List 124
10.5 User-Defined Data 125
10.6 Callbacks 126

11 Using Template Builder 127
11.1 Getting Started 128
11.2 Using Template Builder 128

Part II: Reference Appendices

A Color Object 135

B Display X11 Object 143

C Document Object 147

D DrawStyle Object 163

E Font Object 167

F Font Family Object 171

G Frame Object 175

H Image Object 195

I List Object 201

J Page Object 213

K Printer Object 221

L Print Dialog and Preview Widgets 233

M PtrOption and PtrOptionValue Objects 237

N Style Object 243

O Tab Object 255

P Table Object 259

Border 259
Cell 260
Column 262
Row 263
Table 265

Q Functions and Macros 297

R Data Types 303

S Demos and Examples 309

Index 329

Preface

Introduction ■ *Assumptions*
Typographical Conventions Used in this Manual ■ *Overview of Manual*
Technical Support ■ *Request for Comments*

Introduction

KL Group PageFormatter offers the developer a set of methods and procedures for adding paginated, formatted, flowed-text and image output to UNIX applications. PageFormatter offers support for Western European languages using ISO-Latin-1 fonts.

PageFormatter can be used for the creation of applications that use complex layouts involving columns, "frames," multiple layout styles on one page, the inclusion of automatically generated headers and footers, different fonts, and more.

PageFormatter may be used in conjunction with X/Motif or may be part of a non-X application.

Assumptions

This manual assumes that you are proficient with the C language. C concepts such as "an array of `char *`," and "pointer to a structure" must be understood before beginning to program PageFormatter applications.

Typographical Conventions Used in this Manual

`Typewriter Font`	■ C language source code and examples of file contents. ■ Attribute names and constants. ■ Commands that you enter at a terminal window.
Italic Text	■ Pathnames, filenames, program, methods, and parameter names. ■ New terms as they are introduced, and to emphasize important words. ■ Figure and table titles.

Overview of Manual

Part I describes how to use the PageFormatter.

Chapter 1, "Developing a Simple KL Group PageFormatter Program," provides a hands-on introduction to PageFormatter by giving you the chance to compile and run a simple example program. It should be read by all programmers learning to use PageFormatter.

Chapter 2, "PageFormatter Basics," provides basic information that you need to know before starting to develop applications with PageFormatter.

Chapters 3 through 10 provide programming information for PageFormatter objects.

Chapter 11, "Using Template Builder," explains how to to create page templates using Template Builder.

Part II contains detailed technical reference information in a number of appendices.

Appendices A through P, provide an alphabetically organized list of PageFormatter objects, their attributes, and their methods.

Appendix Q, "Functions and Macros," lists convenience functions and macros.

Appendix R, "Data Types," lists the PageFormatter data structures.

Appendix S, "Demos and Examples," lists the various demos and example programs and source code supplied with PageFormatter to aid in learning.

What's New

Table Support

PageFormatter now supports the creating of tables with the new Table object. Tables can be rendered as part of the flow or in a specific frame. For more information on tables see Tables on page 89.

Drawing Tools

New methods have been added to the Frame object, allowing you to add lines, circles, and polygons to specific frames. A new object, the DrawStyle object, has also been added. The DrawStyle object defines a draw graphic's line weight, line color, and fill. For more information on the new drawing methods, see Drawing Tools on page 60.

Adobe's Acrobat Portable Document Format

PageFormatter now provides a printer type that allows you to output your Document object to Adobe's Acrobat Portable Document Format (PDF). PDF format allows you

to generate compact files readable across various platforms that retain their formatting characteristics.

Image Object, New Image Types, New Image Methods

Images are now defined with the Image object. PageFormatter now supports rendering GIF and JPEG images (as well as XPM and EPS formats). New image methods have been added to the Document and Frame objects to render images using the image object.

Old Method	New Method
KlgPfFlowEmbedImage()	KlgPfFlowImageEmbed()
KlgPfFlowFloatImage()	KlgPfFlowImageFloat()
KlgPfFrameEmbedImage()	KlgPfFrameImageEmbed()
KlgPfFrameFloatImage()	KlgPfFrameImageFloat()
KlgPfFramePlaceImageXY()	KlgPfFrameImagePlaceXY()

For this version, the old methods are still available for compatibility. You should, however, use the new image methods.

Technical Support

KL Group's Customer Service Team provides a variety of support options to suit your needs.

Standard Support

Included free with every PageFormatter license, standard support provides 60 days of unlimited technical support, via email, phone or fax. In addition you can access FAQs and the PageFormatter Knowledgebase on KL Group's web site, and download any minor upgrades or patches, via the Internet.

Gold Support with Subscription

In addition to all the benefits of Standard Support, Gold Support provides unlimited technical support and all major updates shipped to you, with enabling passwords, for one year.

For information on how to obtain Gold Support for your KLGPR product, contact the KL Group Sales Team at (416) 594-1026.

Contacting Technical Support

If you purchased PageFormatter from KL Group, contact KL Group's Customer Service department for technical support; if you purchased PageFormatter from a

reseller, please contact the reseller for technical support. Specific technical support contact information is included with your installation password document.

Request for Comments

We are interested in hearing about how you use PageFormatter, any problems you encounter, or any additional features you would find helpful. The majority of enhancements to KL Group's products are the result of customer requests.

Please write, phone, fax or email your comments to:

KL Group Inc.
260 King Street East
Toronto, Ontario, M5A 1K3 Canada

Phone: (800) 663-4723 or (416) 594-1026
Fax: (416) 594-1919
Email: `pageformatter@klg.com`
`Internet: news://news.klg.com/klg.forum.PageFormatter`

PageFormatter Announcements

Occasionally, we send PageFormatter-related product announcements to our customers using an email list. To add yourself to this mailing list, send email with the word "`subscribe`" in the body of the message to `PageFormatter_announce-request@klg.com`. Visit the KL Group web site at `http://www.klg.com` for more details.

Part I

Using KL PageFormatter

1

Developing a Simple KL Group PageFormatter Program

Setting Your Environment ■ *Compiling and Authorizing a Program*
Basic Flowing And Printing Techniques ■ *Tabs*

This chapter allows you to immediately try out PageFormatter by compiling and running an example program which introduces some fundamental PageFormatter programming concepts. This application flows, renders, and prints a block of text to a PostScript printer.

1.1 Setting Your Environment

1.1.1 KLGPRHOME Environment Variable

Before getting started, be sure your Unix environment is set up correctly. An environment variable called *KLGPRHOME* should be defined as the name of the directory containing the PageFormatter distribution files.

You can determine the current value for *KLGPRHOME* by entering:

```
echo $KLGPRHOME
```

If your *KLGPRHOME* environment variable is not set up correctly, see your system administrator or the *Release and Installation Notes*.

1.1.2 Other Environment Variables

Use PageFormatter's *KlgPfGlobalGet()* and *KlgGlobalSet()* functions to set the file or directory location for printer (.PPD) or font (.AFM and HP TFM) files.

For more information on these variables, see Appendix Q .

1.2 Compiling and Authorizing a Program

If the X and Motif libraries and include files are installed in the standard places (normally */usr/lib*, */usr/include/X11*, and */usr/include/Xm), *simple.c* may be compiled and authorized by entering the following commands:[1]

```
cc simple.c -I$(KLGPRHOME)/include \
   -L$(KLGPRHOME)/lib -o simple \
   -lklgpf -lXm -lXpm -lXt -lX11 -lm
$(KLGPRHOME)/bin/klgpr_auth simple
```

Note: You do not need X and Motif installed to use PageFormatter. See the Makefile, located in *$KLPRHOME/src/pf/examples* for an example of how to build an application with X and Motif.

If compiling from a shell, remove the parenthesis around `KLGPRHOME`. If the libraries and include files are not installed in standard locations, additional use of the *-I* and *-L* compiler flags is required.

klgpr_auth

After the PageFormatter library has been linked into an executable application, it must be authorized with the *klgpr_auth* utility. This utility will check that the machine you are building on is authorized to use PageFormatter, and will enable the application. *klgpr_auth* takes two optional switches, `-v` for verbose messages and `-q` for quiet mode (that is, no messages):

```
klgpr_auth [-q | -v] executable_file
```

Once an end-user application has been enabled by *klgpr_auth*, it may be executed on any machine, anywhere. There are no special runtime requirements for applications built with PageFormatter.

C++ and ANSI C

The PageFormatter include files are compatible with C++ and ANSI C as well as with traditional “Kernighan and Ritchie” C.

1.3 Basic Flowing And Printing Techniques

Figure 1 lists the program *simple.c*. This is a minimal C program that flows some hard-coded text into a Document object and then prints the rendered text using the Printer object.

The simplest way to flow text into a Document object is to pass text to the *KlgPfFlowPrint()* method and allow the flow to progress as defined by the template Page object.

Consider a simple example that will print a single line of text:

1. Compilation options depend on your computer architecture. See the *Makefile* in *$KLGPRHOME/src/pf/examples* for an example of how to compile on your system.

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <KlgPfAll.h>
4. #include <KlgPfPs.h>
5. /*
6. * define a page template, its frames,
7. * the order the main flow moves through
8. * the page's frames, and what page template
9. * to use when the flow transfers to a
10.* subsequent page
11.*/
12.
13.String template_dat = "PageTemplate mypage {\n"
14."   X: 0\n"
15.  "Y: 0\n"
16.  "Width: 8.5\n"
17.  "Height: 11\n"
18.  "Unit: inches\n"
19. /* define a frame on a page */
20.  "Frame header {\n"
21.  "X: 1\n"
22.  "Y: 0.5\n"
23.  "Width: 6.5\n"
24.  "Height: 0.5\n"
25.  "}\n"
26.  "Frame footer {\n"
27.  "X: 1\n"
28.  "Y: 10\n"
29.  "Width: 6.5\n"
30.  "Height: 0.5\n"
31.  "}\n"
32.  "Frame body {\n"
33.  "X: 1.5\n"
34.  "Y: 1.25\n"
35.  "Width: 6\n"
36.  "Height: 6\n"
37.  "}\n"
38. /* define the sequence of frames the flow will use * /
39.  "FlowFrameList: body\n"
40. /* define the template to use when the next page is created*/
41.  "FlowPageTemplate: mypage\n"
42. /* define the template used after a section break*/
43.  "FlowSectionTemplate: mypage\n" "}";
44.
45. int
46. main(int argc, char *argv[])
47. {
48.  FILE *outfile;
49.  KlgPfObject doc, printer;
50.  /* create a printer object */
51.  printer = KlgPfPrinterCreatePS("psprinter");
52.  if (!(outfile = fopen("simple.ps", "w"))) {
53.      printf("\nCannot open file \"simple.ps\" for writing. Exiting...\n\n");
54.       exit(1);
55.  }
```

Figure 1 Simple Flow Example (continued next page)

```
56.  KlgPfPrinterSetOutputType(printer, KlgPfIOStream);
57.  KlgPfPrinterSetStream(printer, outfile);
58.
59.  /* create a document object */
60.  doc = KlgPfDocCreate(printer, "simple");
61.
62.  KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, template_dat);
63.
64.  KlgPfDocBegin(doc);
65.  KlgPfFlowPrint(doc, "I call our world Flatland, not because we “);
66.  KlgPfFlowPrint(doc, "call it so, but to make its nature clearer to you, my  “);
67.  KlgPfFlowPrint(doc, "happy readers, who are privileged to live in Space.");
68.
69.  KlgPfDocEnd(doc);
70.  KlgPfPrinterOutput(printer, doc);
71.  KlgPfDocDestroy(doc);
72.  KlgPfPrinterDestroy(printer);
73.  return(0);
74.  }
```

Figure 1 Flow Example (Continued)

1.3.1 What the Code Does

Between lines 13 and 43, we define a template Page object. A template is used to define the size and location of a page to be rendered, the size and location of frames on the page, and how the flow should move from frame to frame and from page to page. In this case, we've created a template definition for *mypage* that we've stored in `template_dat`. The template could also be defined in a header file and included via `#include`.

On lines 51 and 60, we create a Printer object and a Document object. On line 62, we create a template Page object based on the `template_dat` definition. *KlgPfPageTemplateCreate()* parses the definition, creates a *mypage* template Page object, and adds it to the specified Document object's `PageTemplateList` list. The *KlgPfIOBuffer* parameter indicates the method should parse a template defined in a given string buffer.

KlgPfPageTemplateCreate() also reads all the frames you have defined in the template Page object, creates Frame objects for each one, and adds them to the *mypage* template Page object's `FrameList` list. In this example, we've created three frames: a *body* frame, a *header* frame, and a *footer* frame.

KlgPfDocBegin()

KlgPfDocBegin() on line 64 initializes the first Page object to be rendered based on the first template in the Document object's `PageTemplateList` (which is *mypage*). As well, it initializes the page's Frame objects based on the frames in the Page object's `FrameList` attribute, creating a *header*, *body*, and *footer* frame on the page.

KlgPfFlowPrint()

The string passed to *KlgPfFlowPrint()* on line 65 will be rendered in the first frame defined by the Page object's `FlowFrameList` list. Back on line 62, *KlgPfPageTemplateCreate()* also read the template Page object definition's *FlowFrameList* (on line 39) and added the indicated frames to the Page object's `FlowFrameList` list. In this example, only one of the three frames defined on the page was included in the *FlowFrameList*. The *header* and *footer* frames were not included because normally the text flow does not get rendered to such areas of a page.

The text passed to *KlgPfFlowPrint()* will be rendered in the *body* frame, beginning at the indicated X and Y coordinates (1.5" from the document's left edge and 1.25" from the document's top edge). The text will be rendered on a line no longer than the frame's width (that is, 6.5"). When it reaches the frame's right boundary, PageFormatter continues rendering the text on the second line. This process repeats until:

- It runs out of text, returning `True`; or
- The flow reaches the bottom portion of the frame.

If *KlgPfFlowPrint()* reaches the bottom of the frame and still has text to render, it transfers the flow to the next frame in the Page object's `FlowFrameList` list (which was originally defined by the template Page object definition's *FlowFrameList* line). If there are no more frames in the `FlowFrameList` list, a new page is generated.

The new page is generated based on the template Page object defined by the Page object's `FlowPageTemplate` attribute (which was originally defined by the template Page object definition's *FlowPageTemplate* line and was added to the Page object's `FlowPageTemplate` attribute by the *KlgPfPageTemplateCreate()* method). In the Figure 1 example, we always want the next page to be based on the *mypage* template.

KlgPfDocEnd()

In line 69, we call *KlgPfDocEnd()*. If this were a more complex document, some pages or frames might not be completely rendered until the very end. For example, if we had a frame that displayed the total number of pages in the document, pages with this frame could not be completely rendered until the very end when the page total was known. We would need to call *KlgPfDocEnd()* to signal this.

KlgPfPrinterOutput()

At this point, the document has yet to be actually printed. It has been rendered in memory (or in temporary files). In line 70, we call *KlgPfPrinterOutput()*, passing the fully rendered Document object and the Printer object. *KlgPfPrinterOutput()* actually handles the output to a printer or a screen preview device.

KlgPfDocDestroy() and KlgPfPrinterDestroy()

Finally, we destroy both the Document object (which destroys all its child objects like the Page objects and Frame objects) and the Printer object.

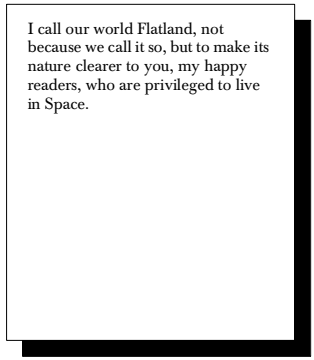


Figure 2 Sample output from Figure 1

1.4 Looking Ahead

The example in this section flows a simple bit of text. PageFormatter's real benefit is its library of simple-to-use methods for rendering text in various font styles, generating tables, aligning text with tabs, and outputting to PostScript printers, PCL-compatible printers, and Acrobat PDF files.

This manual starts with the basics before delving into more powerful features. However, below is an advanced peek at features to be explored in later chapters.

1.4.1 Styles

PageFormatter can render paragraphs, words, and characters using many of the font and paragraph formatting features found in word processors: bold/italics, multiple fonts, centered text, and so forth. You alter the appearance of text by creating or modifying Style objects.

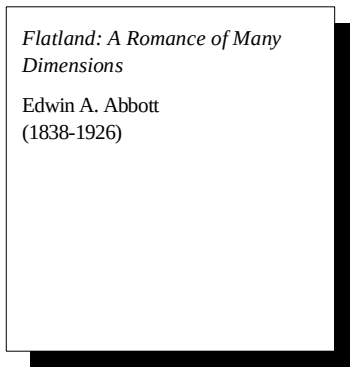
The code below retrieves the Style object PageFormatter is applying to rendered text and modifies its font style.

```
curr_style = KlgPfDocGetCurrentStyle(doc);

KlgPfStyleSetFontStyle(curr_style, KLGPF_FONT_ITALIC);
KlgPfFlowPrint(doc, "Flatland: A Romance of Many Dimensions");
KlgPfFlowLineFeed(doc);

KlgPfStyleSetFontStyle(curr_style, KLGPF_FONT_NORMAL);
KlgPfFlowPrint(doc, "Edwin A. Abbott");
KlgPfFlowLineFeed(doc);
KlgPfFlowPrint(doc, "(1838-1926)");
```

The code would produce the following output:



For more information on styles, see Chapter 6.

1.4.2 Creating a Simple Table

You can create, populate, render, and destroy an entire table with the *KlgPfTableCreateSimple()* method. *KlgPfTableCreateSimple()* is passed the document or current frame you wish to render the table in, an array of column widths, the number of columns in the table, and a 2D array of Strings to be used as cell contents. Each row of the array must have as many entries as there are columns in the table. `NULL` indicates a blank cell.

```
#define NROWS    3
#define NCOLUMNS 3

 int         i;
 static double      columns[] = {1.00, 1.25, 2.25, -1.0};
 static String      rows[][NCOLUMNS] = {{ "Apple", "Orange", "Fructose" },
                                        { "Godzilla", "King Kong", "Barney" },
                                        { "Foil", "Epee", "Big Stick" } };
 String      *cell_data[NROWS + 1];
 KlgPfObject table, current_frame;

  for (i = 0; i < NROWS; i++) {
  cell_data[i] = rows[i];
 }
 cell_data[NROWS] = NULL;

 /* Create the page templates and start the document */
 KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, template_dat);
 KlgPfDocBegin(doc);

 KlgPfFlowPrint(doc, "Simple table example:  this program ");
 KlgPfFlowPrint(doc, "demonstrates construction of a simple table ");
 KlgPfFlowPrint(doc, "using PageFormatter.");

 current_frame = KlgPfDocGetCurrentFrame(doc);

 KlgPfTableCreateSimple(current_frame, columns, NCOLUMNS, cell_data);

 KlgPfDocEnd(doc);
```

If the code was added to a PageFormatter program, you would get the following output:

Simple table example: this program demonstrates construction of a simple table using PageFormatter.

Apple	Orange	Fructose
Godzilla	King Kong	Barney
Foil	Epee	Big Stick

For more information on tables, see Chapter 8.

1.4.3 Tabs

The Tab object provides methods for adding tabs and modifying the tab's type, horizontal position, and appearance.

Creating a Tab Object

A Tab object is created with the *KlgPfTabCreate()* method. You can set attributes like tab fill and tab alignment.

For example:

```
style = KlgPfDocGetCurrentStyle(doc);
new_tab = KlgPfTabCreate(style, 1.0);
KlgPfTabSetFillType(new_tab, KLGPF_TAB_FILL_WITH_ELLIPSES)
```

To use our created tab, you call the *KlgPfFlowTab()* method.

For example, to render:

```
1).........I'm following a left-aligned tab.
```

You could write the following:

```
KlgPfFlowPrint(doc, "1)");
KlgPfFlowTab(doc);
KlgPfFlowPrint(doc, "I'm following a left-aligned tab.");
```

1.5 Proceeding From Here

The following suggestions should help you be productive with PageFormatter as quickly as possible:

- Chapter 2 introduces basic PageFormatter concepts. You should read this chapter to get a good grounding in PageFormatter.
- Chapter 3, Chapter 4, and Chapter 5 look at the Document object, the Page object, and the Frame object, respectively, in greater detail.
- Chapter 6 looks at the Style object, its methods and attributes that allow you to take advantage of many different formatting techniques.
- Advanced formatting using Tab and Color objects as well as incorporating images into your document are covered in Chapter 7.
- Chapter 8 looks at the Table object, its methods and attributes that allow you to create tables.
- The examples and demos installed with PageFormatter are useful learning aids. You can see what PageFormatter can do, and how to do it, by running them and examining their source code. See Demos and Examples on page 309 for more information.

2

PageFormatter Basics

Objects ■ *Attributes and Methods*
Rendering Text with Flow and Frame Methods ■ *Printing*

2.1 Basic Steps for Creating a PageFormatter Document

Below are the basic steps required to format and flow a document.

	Step	See
1.	Create the Printer object.	Printer Object on page 109
2.	Create the Document object.	Document Object on page 27
3.	Change the printer and document attributes as desired.	Printer Object on page 109 and Document Object on page 27
4.	Create the document template.	Page Object on page 37 or Using Template Builder on page 127
5.	Create/Edit Style objects.	Style Object on page 67
6.	Call *KlgPfDocBegin()* to start flow to the document.	Document Object on page 27
7.	Flow text and tables and change styles and attributes as necessary.	Document Object on page 27 and Style Object on page 67
8.	Call *KlgPfDocEnd()* to end flow to the document.	Document Object on page 27
9.	Call *KlgPfPrinterOutput()* to send the document to the specified printer.	Printer Object on page 109
10.	Destroy the objects you created, such as the document object, the printer object, and any color, border, or image objects.	Document Object on page 27, Printer Object on page 109, Advanced Formatting on page 79, and Tables on page 89

2.2 Objects

PageFormatter uses an object-oriented approach. It uses a series of nested objects. The Document object is the top-level object. Page, Frame, Color, Table, and Tab objects are created, defined, and added to the Document object. Text is flowed into the Frame objects. When all text has been flowed into the Document object, it is passed to a Printer object which actually handles the outputting of the document.

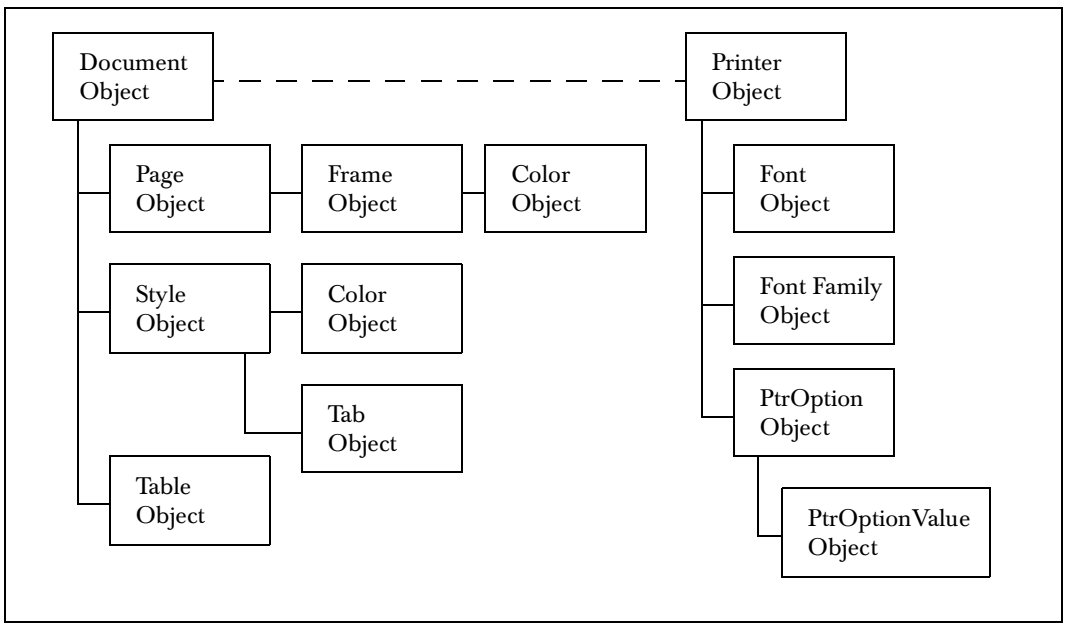


Figure 3 Relationship between objects

2.2.1 Object Overview

To use PageFormatter, you need to understand how the objects are used to create a document and output it. Below is a brief overview of the objects used by PageFormatter. Each object is covered in both a chapter and a reference section.

Object	Description
Document Object	The Document object is the object that ultimately holds the Page and Frame objects into which you “flow” or “render” your text and images. Object attributes specify how the flow should advance through the rendering process. The Document object provides methods for initiating the flow, re-directing the flow, and changing the layout style of the flow.
Page Object	The Page object is a child of the Document object. The Page object’s attributes specify how an actual page should be laid out, what Frame objects should exist on the page, and the frame order of the flow. Rendered Page objects are based on template Page objects. Complex documents may need to define many different template Page objects. For example, one might need to define template Page objects that describe a title page, a table of contents page, a first page, and subsequent body pages.
Frame Object	The Frame object is a child of the Page object. The Frame object’s attributes specify where on the page the text should be rendered. Complex documents may need to define many different Frame objects on a page. For example, you might wish to define a header frame, a body frame, and a footer frame.
Style Object	The Style object is a child of the Document object. You alter the appearance of text by creating or modifying Style objects. A Style object’s attributes identify how the rendered text should appear.
Table Object	The Table object is a child of a Document or Frame object. The Table object provides methods and attributes for creating tables. Using the Table object’s sub-objects (the Row, Column, Cell, and Border objects) you can customize the table’s rows, columns, cells, and borders.
Tab Object	The Tab object is a child of the Style object. The Tab object provides methods for modifying the tab’s type, horizontal position, and appearance (that is, its fill type).
Color Object	The Color object provides methods for the creation of colors. The Color object creates a color by specifying the intensity of Red, Blue, and Green attributes.
Printer Object	The Printer object provides attributes and methods that control how the rendered document is printed on an output device: the number of pages to output, the number of copies, whether copies should be collated, and so on. The Printer object, when created, also generates a list of Font Family and Font objects based on provided printer drivers. Provided printer drivers also supply the Printer object with information about printer options such as what paper sizes are supported (for example “Letter”, “Legal”, “A4”, and so on) and what resolutions are supported. A Printer object needs to be created before a Document object and passed to the Document object. The Document object uses the Printer object’s font information to properly flow the rendered text.
Font Object	The Font object is a child of the Printer object. PageFormatter represents every printer font available as a Font object. Font objects are held in the Printer object’s FontList list attribute. When you create a Printer object, the FontList is generated automatically.

Object	Description
Font Family Object	The Font Family object is a child of the Printer object. The Font Family object provides methods and a structure for grouping together individual fonts into a “font family”.
PtrOption Object	The PtrOption object is a child of the Printer object. The PtrOption object provides attributes and methods for defining printer options (for example, PageSize, Resolution).
PtrOptionValue Object	The PtrOptionValue object provides attributes and methods for defining individual values available for each PtrOption object.

2.2.2 Objects on a Page

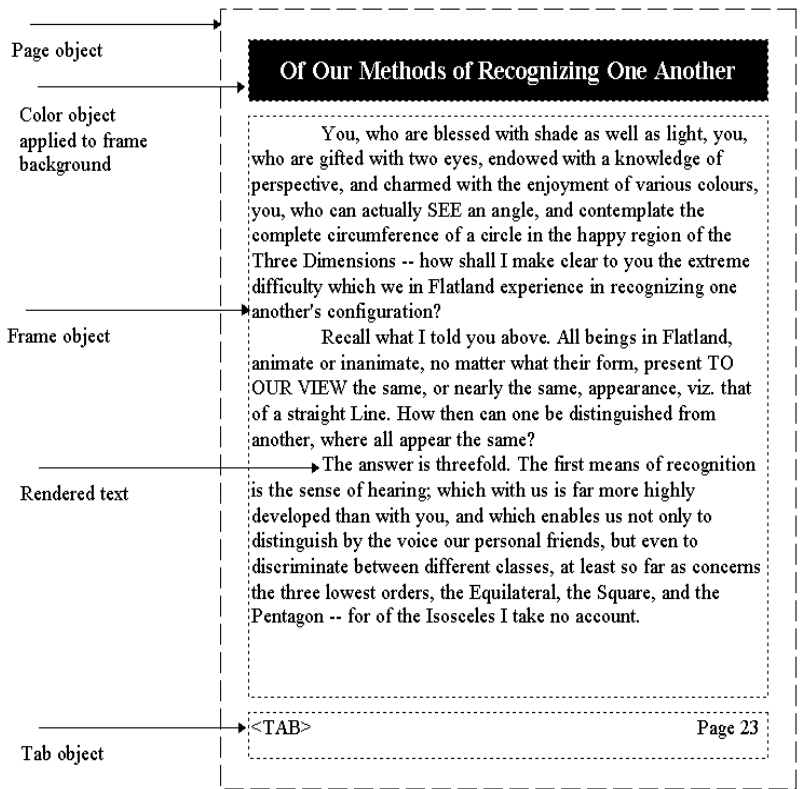


Figure 4 Example of how objects are used to build a page

The sample page in Figure 4 has a Page object which is a child of the overall Document object. The Page object has three child Frame objects (a *header* frame, a

body frame, and a *footer* frame). The *header* Frame object has a child Color object that defines the frame's background as black.

The Document object also has several Style objects:

- One Style object is applied to the rendered text in the *header* frame, setting the font color to white, the font size to 14 points, and the paragraph alignment to center.
- A second Style object has been defined and has been applied to the *body* frame's text, setting the font to 12 points, the paragraph alignment to left, and the first line indent to .5".
- Finally, the Document object has a third Style child object which is being applied to the footer frame. This Style object has a Tab child object defined which, when *KlgPfFrameTab()* is called, right aligns the text at the right margin.

2.3 Attributes and Methods

Each object is comprised of *attributes* and *methods*. A complete list of objects, their attributes, and their methods can be found in Part II of this manual.

2.3.1 Attributes

Object attributes specify, among other things, the appearance and location of your text, images, and frames. For example, the Color object has `Red`, `Green`, and `Blue` component attributes that PageFormatter uses to determine the object's color.

List Attributes

Many objects have more than one child object of the same type. For example, the Page object can have many Frame objects (for example, a *header*, *footer*, and *body* frame). Where an object can have multiple child objects of the same type, they are stored in a List object.

As additional child objects get created, they are added to the parent object's List object. For example, the Page object has a `FrameList` List object that holds all the Frame objects on that page.

For more information on List objects, see Chapter 10 and Appendix I.

2.3.2 Get and Set Methods

Attributes may be set or get using provided methods. For example, the *KlgPfColorSetGreen()* method sets the Color object's `Green` attribute. The *KlgPfColorGetGreen()* method returns the value of the Color object's `Green` attribute.

Most set methods return `True` if the attribute in question is set successfully or `False` if the method fails.

2.3.3 Create and Destroy Methods

Objects are created and destroyed with provided methods. For example, the *KlgPfColorCreate()* method creates a Color object. The *KlgPfColorDestroy()* method frees the memory used by the Color object.

Most object create methods return `NULL` if the method fails.

Destroying Parent Object and Child Objects

Most objects exist as a child of their parent object. When you destroy a child object, it gets removed from its parent object's attribute. For example, a Style object is saved in a Document object's `StyleList` list. When you destroy a Style object, it gets automatically removed from the Document object's `StyleList` list.

When you destroy a parent object, PageFormatter destroys all child List objects automatically. For example, when you destroy the Document object, PageFormatter automatically destroys all its Style objects in its `StyleList` list.

Note: You should not attempt to destroy a parent object and then try to destroy its child List objects, even those objects explicitly created with a create method. This is not true for Color objects, strings and table borders. You are responsible for destroying these.

2.3.4 Units

Many Set methods set a certain page-, frame-, or style-related dimension. For example, a page width, a frame's X/Y position, or paragraph line spacing. Values passed to these methods are interpreted based on the Document's `Unit` attribute. If the Document object's `Unit` attribute is equal to `KLGPF_UNIT_INCHES`, PageFormatter will interpret `KlgPfStyleSetLineSpacing(style, .025)` as setting the Style object's `LineSpacing` attribute to .025".

The Document object's `Unit` attribute is set with the *KlgPfDocSetUnit()* method.

2.4 Rendering Text with Flow and Frame Methods

While the objects define where and how the text gets rendered, PageFormatter's flow methods (for example, *KlgPfFlowPrint(), KlgPfFlowSectionBreak()*) and frame methods (for example, *KlgPfFramePrint()*, *KlgPfFrameLineFeed()*) are the actual methods by which text gets flowed into a page's Frame objects.

2.4.1 Flow Methods

A text flow is initiated with the *KlgPfFlowPrint()* method. The Page object's `FlowFrameList`, `FlowPageTemplate`, `FlowSectionTemplate` attributes control the flow order through frames and pages.

Flow Method	Description
KlgPfFlowColumnFeed()	Advances the flow to the top of the next column. If there is no next column, *KlgPfFlowFrameFeed()* is called.
KlgPfFlowImageEmbed()	Draws the specified image as an inline graphic.
KlgPfFlowImageFloat()	Draws the specified image on its own line.
KlgPfFlowFormFeed()	Generates a new page based on the page template defined by the current Page object's `FlowPageTemplate` attribute. The flow is transferred to the first frame on the new page.
KlgPfFlowFrameFeed()	Generates a frame feed and transfers the flow to the next frame indicated by the Page object's `FlowFrameList` attribute. If there is no next frame, a new page is generated.
KlgPfFlowLineFeed()	Generates a line feed, starting a new paragraph, and transfers the flow to the next line. If there is no more space in the frame, the flow is transferred to the next column in the frame.
KlgPfFlowPrint()	Renders the specified string. If there is no more space in the line, the flow is transferred to the next line in the `column`.
KlgPfFlowSectionBreak()	Generates a section break. *KlgPfFlowSectionBreak()* assigns a new page template based on the Page object's `FlowSectionTemplate` attribute, generates a new page, and transfers the flow to the first frame on the new page.
KlgPfFlowTab()	Generates a tab feed in the current flow.

2.4.2 Frame Flow Methods

The frame flow methods can be used when you want to render text that is not part of the main flow (for example, to specific frames on a page or in a page template).

Frame flow methods cease rendering passed text or images when the flow reaches the bottom of a frame, since they are not part of the main flow.

Frame Method	Description
KlgPfFrameColumnFeed()	Generates a column break and transfers the flow to the next column in the specified frame.
KlgPfFrameImageEmbed()	Draws the specified image as an inline graphic in the specified frame.
KlgPfFrameImage(Float)	Draws the specified image on its own line in the specified frame.
KlgPfFrameLineFeed()	Generates a line feed, starting a new paragraph, and transfers the flow to the next line in the specified frame.
KlgPfFramePrint()	Renders the specified string to the specified frame.
KlgPfFrameTab()	Generates a tab feed in the specified frame flow.

2.5 Printing

Printing is accomplished by passing the rendered Document object and a Printer object to the *KlgPfPrinterOutput()* method. A Printer object can be a PostScript Printer object (created with the *KlgPfPrinterCreatePS()* method), an Adobe Acrobat PDF printer (created with the *KlgPfPrinterCreateAcro()* method), an HP PCL printer (created with the *KlgPfPrinterCreatePCL()* method) or an X11 Printer object (created with the *KlgPfPrinterCreateX11()* method).

2.6 Error Handling

PageFormatter provides the developer a versatile error handling mechanism. If PageFormatter traps an error, it calls its error handler, which by default is *KlgPfError()*, and then returns an appropriate error value (if possible). *KlgPfError()* prints a message to *stderr* indicating the error's severity, location, and description of the problem.

For more information on error handling procedures, see Appendix Q.

2.6.1 Error Levels

PageFormatter has three error levels.

Level	Type of Error
1	Warning
2	Error
3	Fatal error

2.6.2 Setting Your Own Error Handler

You can create your own error handling procedure by defining a procedure that conforms to the following type definition:

```
typedef void (*KlgPfErrorProc)(
   KlgPfErrorStruct    *error,
   XtPointer           user_data
   )
```

You can call *KlgPfErrorSetHandler()* and set your defined error procedure.

2.6.3 Error Structure

On error, PageFormatter passes the following structure to the defined error handler:

```
typedef struct {
   int         error_level; /* read only */
   String      location;    /* read only */
   String      message;     /* read only */
   Boolean     exit;
} KlgPfErrorStruct;
```

error_level is the error level. *location* is an ASCII message indication where the error was returned. *message* is an ASCII description of the problem. *exit*, if `True` (the default), tells PageFormatter to exit on a fatal error. If `False`, PageFormatter will not exit on a fatal error. The latter setting is not advisable but can be used if you want to develop a mechanism to inform the user of the error before exiting.

2.6.4 Setting User Data

On error, PageFormatter will also pass a handle to any defined user data. You can define user data with the *KlgPfErrorSetUserData()* method.

2.7 Method Return Values On Errors

Most PageFormatter methods return a predictable value on error. The table below outlines what you can expect as a returned value should a method fail:

Return Type	Returns On Error
KlgPfObject	NULL
Boolean	False
enum	-1
int	-1
double	-1
XtPointer	NULL

3

Document Object

Creating and Configuring the Document Object ■ *Document Object Attributes*
Controlling the Flow ■ *Callbacks*

3.1 Creating and Configuring the Document Object

The Document object is the object that ultimately holds the Page and Frame objects into which you "flow" or "render" your text and images. Object attributes specify how the flow should advance through the rendering process. A Style list keeps track of all styles that are available with the document. The Document object provides methods for initiating the flow, re-directing the flow, and changing the layout style of the flow.

3.2 Document Object Attributes

Attribute	Type
`CurrentFlow`	int
`CurrentFrame`	KlgPfObject
`CurrentPage`	KlgPfObject
`CurrentStyle`	KlgPfObject
`CurrentDrawStyle`	KlgPfObject
`DrawStyleList`	KlgPfObject
`FlushPolicy`	KlgPfFlushPolicy
`FrameCallback`	KlgPfCallbackRecord
`MemoryMax`	int
`Name`	String
`OutputPolicy`	KlgPfOutputPolicy

Attribute	Type
PageCallback	KlgPfCallbackRecord
PageCount	int
PageList	KlgPfObject
PageNumber	int
PageTemplateList	KlgPfObject
Printer	KlgPfObject
SectionCallback	KlgPfCallbackRecord
StyleList	KlgPfObject
Unit	KlgPfObject

3.2.1 Creating a Document Object

You create a Document object with the *KlgPfDocCreate()* method, for example:

```
doc = KlgPfDocCreate(psprinter, "simple");
```

You need to pass the *KlgPfDocCreate()* method a Printer object because PageFormatter needs to know to what kind of device/printer it is rendering. Various child objects need access to Printer object attributes. For example, the Style object needs to know what fonts are resident in the printer. You also need to pass the Document a unique name attribute as well. If you need to distinguish between two or more Document objects, the name attribute can be retrieved with the *KlgPfDocGetName()* method.

3.2.2 Closing a Document Object

Before passing the Document object to a printer output method, you need to ensure that pages or frames not completely rendered finish being laid out. The *KlgPfDocEnd()* method is used to force the rendering process to finish uncompleted pages. *KlgPfDocEnd()* also finishes the document flow started by *KlgPfDocBegin()*.

3.2.3 Destroying a Document Object

PageFormatter provides a method for freeing memory by destroying the Document object and all its child objects and lists (for example, Page objects, Style objects, and so on). To destroy a Document object, call *KlgPfDocDestroy()*.

3.2.4 Setting Document Object Attributes

The developer should not normally need to call many of the provided methods that set and get attribute values. Most of the Document attributes are set:

- when the Document object is created (for example, the `Printer` attribute is set during the create method);

- when child objects are created (for example, the `PageTemplateList` attribute is set when you parse a page template definition); and
- during the rendering process (for example, the `PageCount` attribute is set as pages are rendered).

The two exceptions are the `PageNumber` attribute which specifies the page number of the page currently being rendered and the `CurrentStyle` attribute. Although these values have default settings, you will often wish to change the values of these attributes.

For a complete list of attributes and methods see Appendix C on page 147.

3.2.5 Setting the Page Number

The Document object has two similar attributes: `PageCount` and `PageNumber`.

PageCount

The `PageCount` attribute specifies the total number of pages created (that is, pages rendered or pages in the process of being rendered). If fourteen pages have been created, `PageCount` has the value `14`.

PageNumber

The `PageNumber` attribute is the page number of the page *currently* being rendered. If fourteen pages have been rendered, is `PageNumber` equal to `14`? Not necessarily. Imagine a document that has rendered a title page, a table of contents page, and the first page of actual body text. At the bottom of the first page of actual body text, you want to put "Page 1". This is, however, technically the third page PageFormatter has rendered. If you attempted to render the page number by getting the `PageCount` attribute with the *KlgPfDocGetPageCount()* method, it would return `3`.

KlgPfDocSetPageNumber()

You should use, instead, the `PageNumber` attribute. The `PageNumber` attribute gets incremented in a manner similar to `PageCount` but can be set by the developer with the *KlgPfDocSetPageNumber()* method, for example:

```
KlgPfDocSetPageNumber(doc, 1);
```

The method is passed a Document object and a new page number. It returns `True` if the `PageNumber` attribute is set successfully.

When you need to actually render the page number, you could then call *KlgPfDocGetPageNumber()* to retrieve the `PageNumber` attribute's value.

Incrementing PageNumber and PageCount

When a new page is generated, `PageNumber` and `PageCount` are both incremented by one. In the preceding example, upon the generation of a fourth page, `PageCount` would have a value of 4 and `PageNumber` would have a value of `2`.

Note: if you were to call *KlgPfDocSetPageNumber()* **before** the creation of the page you would need to set it to the desired page number minus one, since PageFormatter increments `PageCount` and `PageNumber` **after** the initialization of a page.

In a page callback with `reason` equal to `KLGPF_REASON_PAGE_NEW`, `PageCount` and `PageNumber` will already have been incremented. For more information about page callbacks, see page 34.

3.2.6 Setting the Current Style

In brief, the Style object lets you control the appearance of rendered text. (You will find a fuller description of the Style object, its attributes, and methods in Chapter 6 on page 67 and Appendix N on page 243.) Attributes like font, indents, and line spacing are set with the Style object. You can create numerous Style objects like "heading1", "heading2", "blockquote", etc. When you create these Style objects, they get added to the Document object's `StyleList` list attribute.

KlgPfDocSetCurrentStyle()

When text is rendered, it will be rendered according to the Style object designated by the Document object's `CurrentStyle` attribute. In most documents, you will want to change the `CurrentStyle` attribute to handle different formatting needs as they arise. You can do this with the *KlgPfDocSetCurrentStyle()* method.

KlgPfDocStringToStyle()

Normally, when you set the Document object's `CurrentStyle` you will want to set it to an existing style from the Document object's `StyleList` list (for example "heading1"). You can retrieve a Style object by name from the `StyleList` list with the *KlgPfDocStringToStyle()* method.

3.3 Controlling the Flow

You need, at times, for the flow to transfer before it reaches the end of a frame or page. For example, on a title page, you may want only two sentences to appear: a report title in a *Title* frame and the name of the author in an *Author* frame. Immediately after the title text is rendered in the *Title* frame, the flow needs to be transferred to the *Author* frame. The author text is rendered and then the remainder of the text (the actual report text itself) needs to be transferred to a new page.

PageFormatter provides methods for controlling the flow in this manner. Let's look at an example, assuming we are using the page templates defined in Figure 9 on page 45.

```
1     int
2     main(int argc, char *argv[])
3     {
4         KlgPfObject doc, printer;
5
6         printer = KlgPfPrinterCreatePS("psprinter");
7         doc = KlgPfDocCreate(printer, "simple");
8
9         KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, title_template_dat);
10        KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, intropage_template_dat);
11        KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, follow_template_dat);
12
13        KlgPfDocBegin(doc);
14
15        KlgPfFlowPrint(doc, "Flatland: A Romance of Many Dimensions");
16        KlgPfFlowFrameFeed(doc);
17        KlgPfFlowPrint(doc, "Edwin A. Abbott");
18        KlgPfFlowLineFeed(doc);
19        KlgPfFlowPrint(doc, "(1838-1926)");
20        KlgPfFlowFormFeed(doc);
21        KlgPfFlowPrint(doc, "I call our world Flatland, not because we");
22        KlgPfFlowPrint(doc, "call it so, but to make its nature clearer to you, my ");
23        KlgPfFlowPrint(doc, "happy readers, who are privileged to live in Space.");
24
25        KlgPfDocEnd(doc);
26
27        KlgPfPrinterOutput(printer, doc);
28
29        KlgPfDocDestroy(doc);
30        KlgPfPrinterDestroy(printer);
31
32        return(0);
33    }
```

Figure 5 Using Document Flow Methods

If this program is compiled and run, you should get two pages printed out that resemble the output in Figure 6.

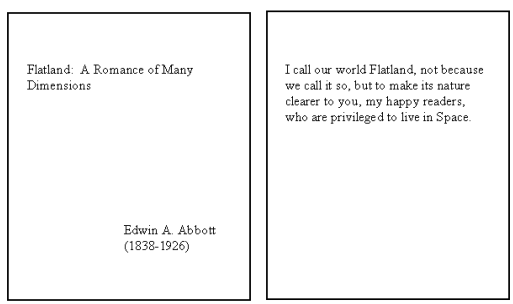


Figure 6 Page layout output

On lines 9 through 11 we add the *title*, *intropage*, and *follow* templates from Figure 9 on page 45 to the Document object. The first Page object created is based on the first template found in the Document object's `PageTemplateList` list.

On line 15, we call *KlgPfFlowPrint()* and the flow then begins in the first frame pointed to by the Page object's `FlowFrameList` list. After rendering the title, the *KlgPfFlowFrameFeed()* method transfers the flow to the next frame indicated by the Page object's `FlowFrameList` attribute.

After rendering the author's name, the *KlgPfFlowLineFeed()* method on line 18 transfers the flow to the next line in the same frame. The author's year of birth and year of death are rendered and then the *KlgPfFlowFormFeed()* method on line 20 generates a new page based on the page template defined by the current Page object's `FlowPageTemplate` attribute. The flow is transferred to the first frame on the new page.

3.3.1 Section Breaks

Section breaks are useful when you want to begin, say, a new chapter and want the first page of the new chapter to use a page layout differing from the one pointed to by the Page object's `FlowPageTemplate` attribute.

When *KlgPfPageTemplateCreate()* parses a page template definition, it adds the name of the template specified in the *FlowSectionTemplate* line to the Page object's `FlowSectionTemplate` attribute. If *KlgPfFlowSectionBreak()* is called, a new Page object is generated and the page is laid out based on the template named by the `FlowSectionTemplate` attribute and not the `FlowPageTemplate`.

3.4 Callbacks

PageFormatter provides a way for your application to be notified when certain events take place, for example when the flow is transferred to a new Frame object or when the text flow generates a new Page object.

3.4.1 Callback Structure

When callback events are generated, PageFormatter passes a callback structure containing the relevant PageFormatter objects as well as a `KlgPfCallbackReason` indentifying what triggered the event.

For example, when PageFormatter generates a new page, it passes the following structure:

```
typedef struct {
     KlgPfCallbackReason    reason;
     KlgPfObject            page;
     KlgPfObject            document;
     KlgPfObject            printer;
} KlgPfPageCallbackStruct
```

Reason

In the case of `KlgPfPageCallbackStruct`, *reason* could be `KLGPF_REASON_PAGE_NEW` (when a new page is generated), `KLGPF_REASON_PAGE_FLOW` (the flow is about to be transferred to a new page), or `KLGPF_REASON_PAGE_END` (when a page has been completely rendered).

3.4.2 Defining Procedures

You can create callback procedures that make use of the information generated by these callbacks and the passed objects. A procedure must conform to PageFormatter's callback type definition:

```
typedef void (*KlgPfCallbackProc)(
   XtPointer   cbs,
   XtPointer   call_data
   );
```

3.4.3 Registering Procedures

After you've created such a procedure, you need to make PageFormatter aware of its existence. Your procedures are registered with the object's callback attribute. The Document, List, and Printer objects provide methods that register procedures you've created to be called when callbacks are generated, and what data, if any, should be passed via the *call_data* parameter. See Appendix C on page 147 for more information.

The Document object provides three attributes to specify callbacks in relation to flow events:

Attribute	Description	Set Function
`FrameCallback`	Specifies a procedure (along with any user data) to call when a frame is created, when the flow is transferred from a frame, and when the flow reaches the end of a frame.	*KlgPfDocSetFrameCallback()*
`PageCallback`	Specifies a procedure (along with any user data) to call when a page is created, when the flow is transferred from a page, and when the flow reaches the end of a page.	*KlgPfDocSetPageCallback()*
`SectionCallback`	Specifies a procedure (along with any user data) to call when *KlgPfFlowSectionBreak()* is called or new page has been generated because of a *KlgPfFlowSectionBreak()* call.	*KlgPfDocSetSectionCallback()*

FrameCallback

The `FrameCallback` attribute specifies a procedure (along with a handle to any relevant data) that will be called when a frame callback is generated. When PageFormatter generates a frame callback, it passes the following structure:

```
typedef struct {
   KlgPfCallbackReason  reason;
   KlgPfObject          frame;
   KlgPfObject          document;
   KlgPfObject          printer;
} KlgPfFrameCallbackStruct;
```

The *reason* member is one of the following enumerated types:

`KLGPF_REASON_FRAME_NEW`	A new frame has been generated.
`KLGPF_REASON_FRAME_FLOW`	The flow is about to be transferred to a new frame. This callback is generated *before* the flow is rendered to the new frame.
`KLGPF_REASON_FRAME_END`	The frame has been completely rendered.

PageCallback

The `PageCallback` attribute specifies a procedure (along with a handle to any relevant data) that will be called when a page callback is generated. When PageFormatter generates a page callback, it passes the following structure:

```
typedef struct {
    KlgPfCallbackReason reason;
    KlgPfObject         page;
    KlgPfObject         document;
    KlgPfObject         printer;
} KlgPfPageCallbackStruct;
```

The *reason* member is one of the following enumerated types:

`KLGPF_REASON_PAGE_NEW`	A new page has been generated.
`KLGPF_REASON_PAGE_FLOW`	The flow is about to be transferred to a new page. This callback is generated *before* text is rendered to the first frame on the new page.
`KLGPF_REASON_PAGE_END`	The page has been completely rendered.

SectionCallback

The `SectionCallback` attribute specifies a procedure (along with a pointer to any relevant data) that will be called when a section callback is generated. When PageFormatter generates a section callback, it passes the following structure:

```
typedef struct {
   KlgPfCallbackReason reason;
   KlgPfObject document;
   KlgPfObject printer;
```

```
} KlgPfSectionCallbackStruct;
```

The *reason* member is one of the following enumerated types:

KLGPF_REASON_SECTION_NEW	A new page has been generated because of a *KlgPfFlowSectionBreak()* call. This callback happens before a KLGPF_REASON_PAGE_NEW callback is generated.
KLGPF_REASON_SECTION_FLOW	The flow is about to be transferred because of a *KlgPfFlowSectionBreak()* call.

3.4.4 Sample Code: Defining and Setting Callback Procedures

In the sample code below, we define a callback procedure that will print the page number on the left side of the header frame when the page number is odd and on the right side when the page number is even.

```
static void
pageCB(XtPointer client_data, XtPointer call_data) {
   char buff[256];
   int page_number;
   KlgPfObject page, style, header;
   KlgPfParagraphAlignment alignment;

   KlgPfPageCallbackStruct *cbs = (KlgPfPageCallbackStruct *)
        client_data;

   if (cbs->reason != KLGPF_REASON_PAGE_NEW) {
        return;
   }

   style = KlgPfDocGetCurrentStyle(cbs->document);
   page = KlgPfDocGetCurrentPage(cbs->document);
   header = KlgPfPageStringToFrame(page, "header");
   alignment = KlgPfStyleGetAlignment(style);
   page_number = KlgPfDocGetPageNumber(cbs->document);

   sprintf(buff, "Page Number : %d", page_number);

   if (page_number % 2 != 0) {
        KlgPfStyleSetAlignment(style, KLGPF_ALIGN_RIGHT);
   } else{
        KlgPfStyleSetAlignment(style, KLGPF_ALIGN_LEFT);
   }

   KlgPfFramePrint(header, style, buff);

   /* restore original value*/
   KlgPfStyleSetAlignment(style, alignment);
}
```

We can then register *pageCB()* with the Document object using the *KlgPfDocSetPageCallback()* method. This sets the document's PageCallback attribute to point to *pageCB()*. The callback should be registered before calling *KlgPfDocBegin()*.

For example:

```
doc = KlgPfDocCreate(printer, "test");
KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, template_dat);
KlgPfDocSetPageCallback(doc, pageCB, (XtPointer)NULL);
KlgPfDocBegin(doc);
```

When PageFormatter generates a new page, it will pass `KlgPfPageCallbackStruct` to *pageCB()* as the first parameter. As noted above, PageFormatter knows to use *pageCB()* because it was registered as the `PageCallback` attribute. We can also use *KlgPfDocSetPageCallback()* to tell PageFormatter to pass user-defined data (`call_data`) to the callback procedure through its second parameter. In this case, however, we don't have any data to pass to *pageCB()*, so we've specified `NULL`. The *pageCB()*/user-defined data combination make up a `KlgPfCallbackRecord`.

4

Page Object

Page Object Attributes ■ *Creating a Template Page Object* ■ *Defining a Template Page Object*
Template Keywords ■ *Parsing the Definition* ■ *Creation with KlgPfPageCreate()*
Getting and Setting Page Attributes ■ *Multiple Page Templates* ■ *Destroying a Page Object*

The Page object is a child of the Document object. The Page object's attributes specify how an actual page should be laid out, what Frame objects should exist on the page, and the frame order of the flow. Rendered Page objects are based on template Page objects. Complex documents need to define many different template Page objects. For example, one might need to define template Page objects that describe a title page, a table of contents page, a first page, and subsequent body pages.

4.1 Page Object Attributes

Attribute	Type
`Color`	KlgPfObject
`FlowFrameList`	KlgPfObject
`FlowPageTemplate`	String
`FlowSectionTemplate`	String
`FrameList`	KlgPfObject
`LayoutStatus`	KlgPfLayoutStatus
`Location`	KlgPfRectangle
`Name`	String
`Type`	KlgPfPageType

4.2 Creating a Template Page Object

Before you begin rendering text to a Page object, PageFormatter needs to know the structure of that page: what frames exist on the page, how the flow should move from frame to frame, and what sort of page it should create after it's filled that page. This is accomplished by defining template Page objects previous to run-time.

PageFormatter uses the same structure for a Page object (that is, a rendered page) and a template Page object (a template upon which actual Page objects are based). Template Page objects can be defined in a string placed in source code (like the one in Figure 1 on page 9) or in a text file. Page Template objects can be created and added to the Document object's `PageTemplateList` list attribute with *KlgPfPageTemplateCreate()*.

4.3 Defining a Template Page Object

A template definition has three specific areas: 1) a definition of the page dimensions; 2) definitions of the frames on the page. 3) flow definitions used when frame, page, and section breaks or flow events occur.

You can also create templates using the GUI-based Template Builder. For more information, see "Using Template Builder" on page 128.

An example string definition is shown in Figure 7. The text file definition format is shown in Figure 8.

```
String variable = "PageTemplate mypage {\n"
"   X: 0\n" /* page definition area */
"   Y: 0\n" /* page definition area */
"   Width: 8.5\n" /* page definition area */
"   Height: 11\n" /* page definition area */
"   Color: white\n" /* page definition area */
"   Unit: inches\n" /* page definition area */
"   Frame body {\n" /* defines a frame on the page */
"       X:1\\n"
"       Y:0.5\n"
"       Width: 6.5\n"
"       Height: 10\n"
"       Color: white\n"
"       ColumnCount: 2\n"
"       ColumnSpacing: .5\n"
"       BorderWidth: .02\n"
"       BorderType: plain\n"
"       BorderColor: black\n"
"   }\n"
"   FlowFrameList: body\n"
"   FlowPageTemplate: mypage\n"
"   FlowSectionTemplate: mypage\n"
"}";
```

Figure 7 An example string definition of a template Page object

```
PageTemplate [String name] {
   X: [A double value]
   Y: [A double value]
   Width: [A double value]
   Height: [A double value]
   Color: [a color string or hex RGB value]
   Unit: [inches, centimeters, or points]
   Frame [String name] {
       X: [A double value]
       Y: [A double value]
       Width: [A double value]
       Height: [A double value]
       Color: [a color string]
       ColumnCount: [An int value]
       ColumnSpacing: [An double value]
       BorderWidth: [a double value]
       BorderType: [plain or none]
       BorderColor: [a color string]
       MarginLeft: [a double value]
       MarginRight: [a double value]
       MarginTop: [a double value]
       MarginBottom: [a double value]
   }
   FlowFrameList: [String name of first frame],
       [Next frame],[Next frame]
   FlowPageTemplate: [string name of template]
   FlowSectionTemplate: [string name of template]

}
```

Figure 8 Syntax for defining a page template in a file

4.4 Template Keywords

PageTemplate [String name]

The string name by which the page Template object is identified on the Document object's `PageTemplateList`.

X: [A double value]/Y: [A double value]

When used in the page definition area, the page's beginning x and y coordinates are relative to the upper-left edge of the Document object. When used in a frame definition area, the x and y coordinates are relative to the Page object. Values are interpreted based on the `Unit` keyword.

Width: [A double value]/Height: [A double value]

The width and height of the page or frame. Values are interpreted based on the `Unit` keyword.

Color: [a color string or hex RGB value]

A predefined color string (for example "Purple"). You may also define it as a hex RGB value (for example, in a valid format like #rgb, #rrggbb, #rrrgggbbb, and so on.) For more information on color and the Color object see Appendix A on page 135. The color definition indicates the background color of the page or frame. This line is optional. If a color definition is absent, PageFormatter defaults to a white background.

Unit [inches | centimeters | points]

This can be "inches", "centimeters", or "points". If this setting is different from that Document object's Unit attribute, it will only affect how PageFormatter interprets the values given for this page and only frames it contains. Once this page is read into PageFormatter, its dimensions are converted into Document units, and document units should be used when manipulating the page programmatically.

Frame [string name]

The string name should be a unique name (for example, "body", "header", "footer"). This is the string used to identify the frame in the *FlowFrameList* keyword and the Page object's `FrameList` and `FlowFrameList` lists. You may define any number of frames using the frame definition syntax. (See Figure 9 on page 45 for an example of multiple frame definitions.) When the template definition is parsed by *KlgPfPageTemplateCreate()*, all the frames named in the definition are added to the Page object's `FrameList` list attribute.

ColumnCount: [an int value]

The number of columns in the frame. If this argument is left out, PageFormatter will default to `1`.

ColumnSpacing: [a double value]

The amount of white space between columns. If this argument is left out, PageFormatter will default to `0.25` inches.

BorderWidth: [a double value]

Frames can be rendered with a surrounding border. *BorderWidth* indicates the thickness of the border. If this argument is left out, PageFormatter will default to `0`.

BorderType: [plain | none]

PageFormatter supports two options: "plain" or "none." "Plain" is a single ruling border type. "None" will create no border, even if a width and color are set. If this keyword is not included, PageFormatter will assume "none."

BorderColor: [a color string]

Specifies the color of the border using a predefined color string (for example "Black", "LightGrey", and so on).

MarginLeft: [a double value], MarginRight: [a double value], MarginTop: [a double value], MarginBottom: [a double value]

Specifies the amount of internal white space to leave in the frame.

FlowFrameList: [string name of first frame],[Next frame],[Next frame]

A list of frame names defined in the current page definition (for example, `body,body2,body3`). This list specifies the flow's frame order. When the rendered text reaches the bottom of a frame or when the *KlgPfFlowFrameFeed()* method is called, the flow will be transferred to the next frame on the list. When the rendered text reaches the bottom of the last frame or when the *KlgPfFlowFrameFeed()* method is called in the last frame, a new page is generated.

FlowPageTemplate: [string name of template]

When a new page is generated, the new page layout will be created based on the template specified by this keyword.

FlowSectionTemplate: [string name of template]

When a new page is generated by calling *KlgPfFlowSectionBreak()*, the new page layout will be created based on the template specified by this keyword.

4.4.1 Setting Styles in a Template

Like a Page or Frame object, you can also define Style objects in a template. You do not need to define all keywords. PageFormatter will assign the defaults for undefined keywords.

```
Style [String name] {
                color: [a color string or hex RGB value]
                fontfamily: [a font family name string]
                font: [a font name string]
                fontsize: [an int value]
                fontstyle: [normal | italic | bold | bolditalic]
                alignment: [left | center | right | justify]
                firstlineindent: [a double value]
                leftindent: [a double value]
                rightindent: [a double value]
                linespacing: [a double value]
                textposition: [baseline | superscript | subscript]
                underline: [true | false]

}
```

Style [String name]

The string name by which the Style object is identified in the Document object's `StyleList`.

color: [a color string or hex RGB value]

A predefined color string (for example "Purple"). You may also define it as a hex RGB value (for example, in a valid format like #rgb, #rrggbb, #rrrgggbbb, and so on.) For more information on color and the Color object see Appendix A on page 135. The color definition indicates the color for drawing text. This line is optional. If a color definition is absent, PageFormatter defaults to a black background.

FontFamily: [a font family name string]

A font family (for example, "Times") to be used when rendering text.

FontSize: [an int value]

A font size, in points, to be used when rendering text.

FontStyle: [normal | italic | bold | bolditalic]

A font style to be used when rendering text.

Alignment: [left | center | right | justify]

The paragraph alignment.

FirstLineIndent: [a double value]

The amount, in document units, the text in the first line of a paragraph is rendered from the left side of the current column.

LeftIndent: [a double value]

The amount, in document units, paragraph text is rendered from the left side of the current column. This attribute affects all lines in the paragraph, except the first line.

RightIndent: [a double value]

The amount, in document units, paragraph text is rendered from the right side of the frame or column. This attribute affects all lines in the paragraph, including the first line.

LineSpacing: [a double value]

The amount, in document units, paragraph lines are spaced apart.

TextPosition: [baseline | superscript | subscript]

Specifies whether the text should be rendered as regular "baseline" text, a subscript, or a superscript.

Underline: [true | false]

Flag indicating whether to render text with an underscore.

4.5 Parsing the Definition

Templates, whether defined as strings or defined in text files, need to be parsed and added to the specified Document object's `PageTemplateList`. The Page template data can be read in and parsed with *KlgPfPageTemplateCreate()*.

For example:

```
if (argc == 2) {
   KlgPfPageTemplateCreate(doc, KlgPfIOFilename, argv[1]);
}
else {
   KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, template_dat);
}
```

The method is passed a Document object, a `KlgPfIOType` parameter, and the actual data in the form of a string buffer, a filename, or an open input stream. To read a template in from a file, the method's *io_type* parameter should be `KlgPfIOFilename`. The *data* parameter should be the file name. To create a template from a string, the *io_type* parameter should be `KlgPfIOBuffer`. The *data* parameter should be the string buffer's name.

4.6 Creation with KlgPfPageCreate()

Page objects are automatically created by *KlgPfFlowPrint()* when the flow reaches the end of the last frame or following a flow call that requires a new page (for example, *KlgPfFlowSectionBreak()*). Page objects are, however, ultimately created with the *KlgPfPageCreate()* method, for example:

```
KlgPfPageCreate(doc, "cover", KLGPF_PAGE_ACTUAL);
```

KlgPfPageCreate() is passed the parent Document object, a name attribute, and a type constant. When creating an actual Page object, the type constant should be `KLGPF_PAGE_ACTUAL`. Template Page objects are Page objects but they are flagged as `KLGPF_PAGE_TEMPLATE` (to differentiate them from Page objects being used at render time) and stored in the Document's `PageTemplateList` list.

4.7 Getting and Setting Page Attributes

Like the Document object, you should not normally need to call many of the provided methods that set and get attribute values (a complete list of these methods is available in Appendix J on page 213). You will usually want to set most of the Page attributes ahead of time using the *KlgPfPageTemplateCreate()* method. Other attributes, such as `LayoutStatus`, are set during the render process.

4.8 Multiple Page Templates

In the example in Figure 1 on page 9, we defined only one page template. More complex documents require different layout styles, hence more than one page template. You might, for example, need to define and call on templates to handle a title page, a first page that lacks header and footer frames, and subsequent pages that have the header and footer frames.

4.8.1 Defining Multiple Page Templates in Code

The code sample below demonstrates how to define multiple templates in your code.

```
String title_template_dat = "PageTemplate title {\n"
"  X: 0\n"
"  Y: 0\n"
"  Width: 8.5\n"
"  Height: 11\n"
"  Unit: inches\n"
"  Frame titleframe {\n" /* report title goes here */
"      X: 4\n"
"      Y: 2.25\n"
"      Width: 4\n"
"      Height: 2\n"
"  }\n"
"  Frame authorframe {\n" /* the name of the report author */
"      X: 6\n"
"      Y: 4.25\n"
"      Width: 2\n"
"      Height: 2\n"
"  }\n"
"  FlowFrameList: titleframe,authorframe \n"
"  FlowPageTemplate: intropage\n"
"  FlowSectionTemplate: intropage\n"/* the template used
                                      after a section break */
"}";
String intropage_template_dat = "PageTemplate intropage{\n"
"    X: 0\n"
"    Y: 0\n"
"    Width: 8.5\n"
"    Height: 11\n"
"    Unit: inches\n"
"    Frame body {\n" /* no header & footer on first page */
"        X: 1\n"
"        Y: 1.25\n"
"        Width: 6\n"
"        Height: 6\n"
"    }\n"
"    FlowFrameList: body\n" /* how the flow should move on the page */
"    FlowPageTemplate: follow\n" /* the next page template to use */
"    FlowSectionTemplate: intropage\n" /* the template used after a
                                      section break */
"}";
```

Figure 9 Example of Multiple Page Template Definitions (continued on next page)

```
String follow_template_dat = "PageTemplate follow {\n"
"    X: 0\n"
"    Y: 0\n"
"    Width: 8.5\n"
"    Height: 11\n"
"    Unit: inches\n"
"    Frame header {\n"
"        X: 1\n"
"        Y: 0.5\n"
"        Width: 6.5\n"
"        Height: 0.5\n"
"    }\n"
"    Frame footer {\n"
"        X: 1\n"
"        Y: 10\n"
"        Width: 6.5\n"
"        Height: 0.5\n"
"    }\n"
"    Frame body {\n"
"        X: 1.5\n"
"        Y: 1.25\n"
"        Width: 6\n"
"        Height: 6\n"
"    }\n"
"    FlowFrameList: body\n" /* how the flow should move on the page */
"    FlowPageTemplate: follow\n" /* the next page template to use */
"    FlowSectionTemplate: intropage\n" /* the template used after a
                                          section break */
"}";
```

Figure 9 continued from previous page

4.8.2 Creating Multiple Templates

Once you've created multiple templates, like those in Figure 9, you can then call *KlgPfPageTemplateCreate()* to create each template Page object and have it added to the Document `PageTemplateList` list. The first template you pass should be the first template you want the main flow to begin with when calling *KlgPfDocBegin()*. For example:

```
KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, title_template_dat);
KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, intropage_template_dat);
KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, follow_template_dat);
```

When text is passed to the *KlgPfFlowPrint()* method, the text will first be rendered to a Page object based on the *title* template (since the *title* template was the first template added to the Document object's `PageTemplateList` list attribute). Once the title page's *titleframe* and *authorframe* frames are filled, the flow is transferred to the second page.

The second page gets generated based on the page template designated by the first Page object's `FlowPageTemplate` attribute (which was originally defined by the *title* template definition's *FlowPageTemplate* line). In this case, it is the *intropage* template.

The new Page object's `FlowPageTemplate` attribute gets set to *follow* (since the *intropage* template definition's *FlowPageTemplate* line indicates *follow*) and the flow gets rendered to the first (and in this case only) frame in the second page's `FlowFrameList` list attribute. When the second page's frame is filled, the flow is transferred to the third page.

The third page is generated based on the previous Page object's `FlowPageTemplate` attribute (which was *follow*). The flow is transferred to the first frame in the third page's `FlowFrameList` list attribute, which is the *body* frame. (Although there are three frames on this page (*header*, *body*, and *footer*), the *follow* template definition specified *FlowFrameList* as "`FlowFrameList: body\n`".)

4.9 Destroying a Page Object

When you destroy a Document object, PageFormatter will automatically destroy created Page objects. If the need arises, however, you can destroy Page objects with the *KlgPfPageDestroy()* method.

4.9.1 FlushPolicy

You can have PageFormatter destroy Page objects after they are physically printed, freeing memory immediately, by setting the Document object's `FlushPolicy` attribute to `KLGPF_FLUSH_ON_OUTPUT`. To set the FlushPolicy, call *KlgPfDocSetFlushPolicy()*. You can pass it one of the following values:

`KLGPF_FLUSH_NEVER`	The Page objects will not be destroyed until the Document object is destroyed.
`KLGPF_FLUSH_ON_OUTPUT`	A Page object is destroyed after it has been printed.

5

Frame Object

Frame Object Attributes ■ *Creating a Frame Object* ■ *Setting Frame Dimensions*
Getting and Setting Other Frame Attributes ■ *Frame Borders*
Columns ■ *Multiple Frames on a Page* ■ *Header and Footer Frames*
Page Number Macros ■ *Drawing Tools*

The Frame object is a child of the Page object. The Frame object's attributes specify where on the page the text should be rendered. Complex documents need to define many different Frame objects on a page. For example, you might wish to define a header frame, a body frame, and a footer frame.

5.1 Frame Object Attributes

Attribute	Type
BorderColor	KlgPfObject
BorderType	KlgPfBorderType
BorderWidth	double
Color	KlgPfObject
ColumnCount	int
ColumnSpacing	double
ColumnWidth	double
LayoutStatus	KlgPfLayoutStatus
Location	KlgPfRectangle
MarginBottom	double
MarginLeft	double
MarginRight	double
MarginTop	double

Attribute	Type
Name	String

5.2 Creating a Frame Object

```
figure_frame = KlgPfFrameCreate(page, "figure");

KlgPfFrameSetX(figure_frame, 3.0);
KlgPfFrameSetY(figure_frame, 6.0);
KlgPfFrameSetWidth(figure_frame, 1.5);
KlgPfFrameSetHeight(figure_frame, 1.5);

KlgPfFrameSetBorderType(figure_frame, KLGPF_BORDER_PLAIN);
KlgPfFrameSetBorderWidth(figure_frame, 0.05);
KlgPfFrameSetBorderColor(figure_frame, KlgPfColorCreateFromString("Gray46"));

KlgPfFrameSetMarginRight(figure_frame, 0.10);
KlgPfFrameSetMarginLeft(figure_frame, 0.10);
KlgPfFrameSetMarginTop(figure_frame, 0.10);
KlgPfFrameSetMarginBottom(figure_frame, 0.10);

KlgPfFrameSetColumnCount(figure_frame, 1);
KlgPfFrameSetColumnSpacing(figure_frame, 0.10);
```

Figure 10 Creating a frame

Frames are usually defined ahead of time in a page template definition. Frame objects are ultimately created with the *KlgPfFrameCreate()* method.

In Figure 10, a frame is created with *KlgPfFrameCreate()*. The *KlgPfFrameCreate()* method is passed the frame's parent Page object and a name string for the frame (each frame on the page must have a unique name). It returns a Frame object.

5.3 Setting Frame Dimensions

You will not normally need to explicitly call *KlgPfFrameCreate()* to create frame objects. Frame objects, if defined in a template Page object, get created automatically when a new Page object is created.

There may be instances where you need to create a specific frame on a specific page. After calling *KlgPfFrameCreate()* to create the basic frame, you need to specify its location (via X and Y coordinates) relative to the Page object, as well as width and height values.

5.3.1 X coordinate

To set a frame's `X` coordinate, use the *KlgPfFrameSetX()* method. The *KlgPfFrameSetX()* method is passed a Frame object and an X coordinate indicating, in units defined by the Document object's `Unit` attribute (for example "Inches"), how far from the Page's left boundary the frame should be rendered.

5.3.2 Y coordinate

To set a frame's `Y` coordinate, use the *KlgPfFrameSetY()* method. The *KlgPfFrameSetY()* method is passed a Frame object and a Y coordinate indicating, in units defined by the Document object's `Unit` attribute (for example "Inches"), how far from the Page's top boundary the frame should be rendered.

5.3.3 Width

To set a frame's `Width`, use the *KlgPfFrameSetWidth()* method. The *KlgPfFrameSetWidth()* method is passed a Frame object and a width value, in units defined by the Document object's `Unit` attribute (for example "Inches").

5.3.4 Height

To set a frame's `Height`, use the *KlgPfFrameSetHeight()* method. The *KlgPfFrameSetHeight()* method is passed a Frame object and a height value, in units defined by the Document object's `Unit` attribute (for example "Inches").

5.4 Getting and Setting Other Frame Attributes

Most frame attributes have set and get methods. For a complete list of Frame methods, see Appendix G on page 175. Most frames should be defined ahead of time in a page template definition. While frame attributes may not normally need to be set by the developer in simple applications, there are cases where you may want to change borders and columns for a specific page. For example, your body frame normally renders text in two columns but you may want to force a new page, format a table of values in a single column, and place a ruling border around the frame to create a page that looks more like a figure.

Note: You should set frame attributes *before* text gets rendered to a frame. Setting frame attributes during rendering can have unexpected results.

5.5 Frame Borders

You can add a border to a frame with the *KlgPfFrameSetBorderType()*, *KlgPfFrameSetBorderWidth()*, and *KlgPfFrameSetBorderColor()* methods.

5.5.1 Border Type

KlgPfFrameSetBorderType() is used to set a Frame object's `BorderType` attribute, which specifies whether the frame border should be displayed or not. *KlgPfFrameSetBorderType()* is passed a Frame object and a border type:

`KLGPF_BORDER_NONE`	No border
`KLGPF_BORDER_PLAIN`	A single, solid border

Note: The method is not passed a Boolean to toggle borders on or off as future versions may support different border types.

5.5.2 Border Width

KlgPfFrameSetBorderWidth() is used to specify a Frame object's `BorderWidth` attribute. *KlgPfFrameSetBorderWidth()* is passed the appropriate Frame object and a width value for its border.

The border is rendered entirely inside the frame. For example, if you set a frame with a 0.05" border, the border will occupy in total 0.10" of the frame's internal width and 0.10" of the frame's internal height. Rendering the border internally ensures that the frame's left-most and upper-most boundaries always exist exactly where specified by the frame's `X` and `Y` attributes.

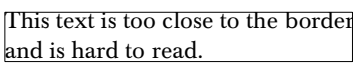
Note: if `BorderWidth` is equal to `0` and `BorderType` is equal to `KLGPF_BORDER_PLAIN`, then a line of the smallest possible width is drawn around the frame.

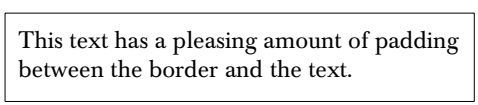
5.5.3 Border Color

When a frame is created, its `BorderColor` attribute is set to "Black" by default. To change this attribute, call *KlgPfFrameSetBorderColor()*. The method is passed the appropriate Frame object and a color object.

For more information on creating a Color object, see Section 7.2 on page 82.

5.5.4 Internal Frame Margins





It is stylistically proper to include some internal white space when using a visible border. A small amount of white space keeps text at a comfortable distance from the border. By default, PageFormatter creates an internal margin of white space equal to

BorderWidth. Alternatively, you can specify "padding" by setting a Frame object's internal MarginRight, MarginLeft, MarginTop, and MarginBottom attributes using the *KlgPfFrameSetMarginRight(), KlgPfFrameSetMarginLeft(), KlgPfFrameSetMarginTop*(), or *KlgPfFrameSetMarginBottom*() methods.

These set methods are passed the appropriate Frame object and the amount of white space, in document units.

Margin settings are measured from the frame's outer edge, not the border's inner edge. If you set a 5 point border and 2 point margins, PageFormatter will overwrite 3 points of border. You should set margins equal to or greater than BorderWidth.

5.6 Columns

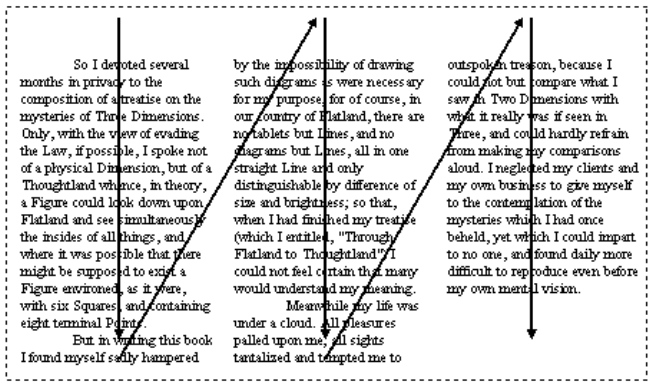


Figure 11 Text flow in a frame with multiple columns.

PageFormatter supports the rendering of text in multiple columns. Columns are created in a frame by setting a Frame object's ColumnCount and ColumnSpacing attributes.

5.6.1 Column Count

The Frame object's ColumnCount attribute specifies the number of columns that exist in the frame. If you need to set this attribute, call *KlgPfFrameSetColumnCount()*. The method is passed the appropriate Frame object and the number of columns.

5.6.2 Column Spacing

Columns should be separated by a comfortable amount of white space. By default PageFormatter sets the Frame object's ColumnSpacing attribute to 0.25. To narrow or enlarge this space, call the *KlgPfFrameSetColumnSpacing()* method. The method is passed the appropriate Frame object and the amount of white space, in document units.

5.6.3 Column Width

Unlike other column-related Frame attributes, there is no set method for the Frame object's `ColumnWidth` attribute. Column width is *calculated* based on `ColumnCount`, the frame width, `BorderWidth`, the frame's internal `MarginLeft/MarginRight`, and `ColumnSpacing`. Changing any of those values results in the automatic recalculation of `ColumnWidth`. Note: if `BorderType` is `KLGPF_BORDER_NONE`, the column width calculation will ignore the `BorderWidth` setting.

Columns are always of equal width.

While there is no set method, you can call *KlgPfFrameGetColumnWidth()* to get the value of the Frame object's `ColumnWidth` attribute. The method is passed the appropriate Frame object. *KlgPfFrameGetColumnWidth()* returns the value of the Frame object's `ColumnWidth` attribute.

5.7 Multiple Frames on a Page

In the page template definition in Figure 1 on page 9, we could have defined more than one body frame. For example:

```
"   Frame body {\n" /* defines a frame on a page */
"       X: 1.5\n"
"       Y: 1.25\n"
"       Width: 6.5\n"
"       Height: 3\n"
"       }\n
"   Frame body2 {\n" /* defines a 2nd frame on the same page */
"       X: 1.5\n"
"       Y: 4.25\n"
"       Width: 6.5\n"
"       Height: 3\n"
"       }\n
```

The *FlowFrameList* line could then be written as:

```
"   FlowFrameList: body,body2\n"
```

When the template definition above is parsed by *KlgPfPageTemplateCreate()*, the *body* frame and the *body2* frame are added to the Page object's `FlowFrameList` list attribute. If the text passed to *KlgPfFlowPrint()* is long (say several pages long), it gets rendered initially in the *body* frame. When it reaches the bottom of the *body* frame, the flow is transferred to the *body2* frame at the indicated *X* and *Y* coordinates. When the flow reaches the bottom of the *body2* frame, a new page is generated. Assuming no changes were made to the page template definition's *FlowPageTemplate* line, the flow continues onto a new page's *body* frame.

5.8 Header and Footer Frames

PageFormatter lets you render text that repeats on every page (for example, a report title in a header frame and a page number in a footer frame). Figure 1 on page 9

created a header and footer frame but they were not included as part of the flow as defined by the `FlowFrameList` list. PageFormatter lets you write to frames that are not part of the main flow with the *KlgPfFramePrint()* method. Further, *KlgPfFramePrint()* allows you to write to either a page template's frame or a frame on a rendered page. Any text you render to a page template's frame will appear on every rendered page that uses that template.

5.8.1 Writing to a Template Frame

```
1  String template_dat = "PageTemplate mypage {\n"
2  "   X: 0\n"
3  "   Y: 0\n"
4  "   Width: 8.5\n"
5  "   Height: 11\n"
6  "   Unit: inches\n"
7  "   Frame header {\n"
8  "       X: 1\n"
9  "       Y: 0.5\n"
10 "       Width: 6.5\n"
11 "       Height: 0.5\n"
12 "   }\n"
13 "   Frame footer {\n"
14 "       X: 1\n"
15 "       Y: 10\n"
16 "       Width: 6.5\n"
17 "       Height: 0.5\n"
18 "   }\n"
19 "   Frame body {\n"
20 "       X: 1.5\n"
21 "       Y: 1.25\n"
22 "       Width: 6.5\n"
23 "       Height: 6\n"
24 "   }\n"
25 "   FlowFrameList: body\n"
26 "   FlowPageTemplate: mypage\n"
27 "   FlowSectionTemplate: mypage\n"
28    "};
```

Figure 12 Writing to a Template Page Frame (continued on next page)

```
29 int
30 main(int argc, char *argv[])
31 {
32     KlgPfObject doc, psprinter, style, template,
          header_frame, footer_frame;
33
34     psprinter = KlgPfPrinterCreatePS("postscript_printer");
35     doc = KlgPfDocCreate(psprinter, "simple");
36
37     KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, template_dat);
38
39     /*
40      * Print into the header and footer
41      */
42
43     style = KlgPfDocStringToStyle(doc, "HeaderFooter");
44     template = KlgPfDocStringToTemplate(doc, "mypage");
45     header_frame = KlgPfPageStringToFrame(template, "header");
46     footer_frame = KlgPfPageStringToFrame(template, "footer");
47
48     KlgPfFramePrint(header_frame, style, "Flatland");
49     KlgPfFrameTab(header_frame, style);
50     KlgPfFrameTab(header_frame, style);
51     KlgPfFramePrint(header_frame, style, "by Edwin A. Abbott");
52

53     KlgPfFrameTab(footer_frame, style);
54     KlgPfFrameTab(footer_frame, style);
55     KlgPfFramePrint(footer_frame, style, "Page @P");
56
57     KlgPfDocBegin(doc);
58
59     KlgPfFlowPrint(doc, "I call our world Flatland, not because ");
       KlgPfFlowPrint(doc, "we call it so, but to make its nature clearer to you, ");
       KlgPfFlowPrint(doc, "my happy readers, who are privileged to live in Space.");
60     KlgPfFlowFormFeed(doc);
61
62     KlgPfFlowPrint(doc, "The greatest length or breadth of a
           full grown inhabitant of Flatland may be estimated at
           about eleven of your inches.");
63
64     KlgPfDocEnd(doc);
65
66     KlgPfPrinterOutput(psprinter, doc);
67
68     KlgPfDocDestroy(doc);
69     KlgPfPrinterDestroy(psprinter);
70
71     return(0);
72 }
```

Figure 12 continued

KlgPfDocCreate() Creates Default Styles

On lines 34 and 35, the program creates a Document and Printer object. The *KlgPfDocCreate()* method populates its `StyleList` attribute with several default Style objects, including a default header/footer Style object with the `Name` attribute "HeaderFooter".

Retrieving Style, Template, and Frame Objects

On line 43, the program calls *KlgPfDocStringToStyle()* to get the header/footer style by its `Name` attribute. *KlgPfDocStringToTemplate()* is called after that to get the "mypage" template that was parsed and added to the Document object by the *KlgPfPageTemplateCreate()* method.

On line 45 and 46, the "mypage" header and footer frames are retrieved from the Page object's `FrameList` by passing the *KlgPfPageStringToFrame()* method the "mypage" template object and the `Name` attribute of the desired Frame objects.

KlgPfFramePrint()

On line 48 and subsequent lines, the *KlgPfFramePrint()* method is used to write text directly into the frame by passing it the Frame object retrieved with *KlgPfPageStringToFrame()* and the desired text.

You will notice the function is also passed a Style object, whereas the *KlgPfFlowPrint()* method is not passed a Style object. Why? The *KlgPfFlowPrint()* method gets passed a Document object which has a `CurrentStyle` attribute and renders the passed text according to the specified style. In the case of *KlgPfPageStringToFrame()*, you are rendering directly to a Frame object which has no attribute that indicates a current style. Hence you have to pass it a specific style. In this case, the "HeaderFooter" style.

KlgPfFrameTab()

On lines 49, 50, 53, and 54 the program calls the *KlgPfFrameTab()* method. *KlgPfFrameTab()* operates similar to *KlgPfFlowTab()* except you have to pass it a Style object (since the Style object contains the list of tab stops in its `TabList` attribute). PageFormatter's default "HeaderFooter" style sets a center tab at 3.25" and a right-aligned tab at 6.5".

In the code sample in Figure 12, we want the author's name to appear right aligned so the program calls *KlgPfFrameTab()* twice. Once to advance to the center tab and a second time to position the text "by Edwin A. Abbott" flush right. The text in the footer frame is also positioned flush right by calling *KlgPfFrameTab()* twice.

Flow Once

Whatever text you write into a template frame with *KlgPfFramePrint()* will always be rendered whenever an actual page based on that template is generated. If the code in Figure 12 was compiled and executed, it would produce the following output:

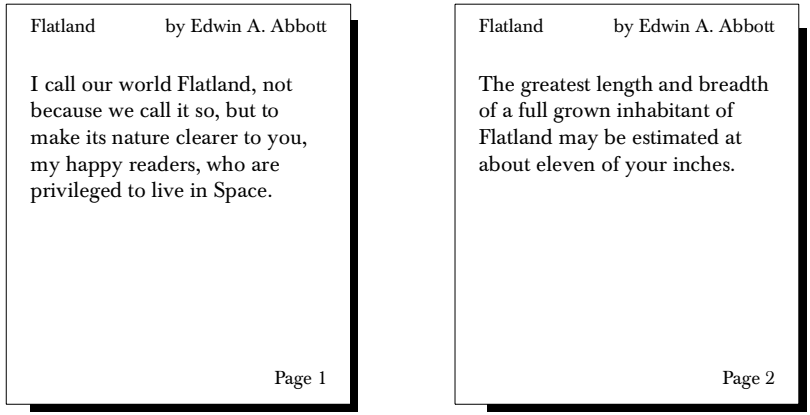


Figure 13 Output generated by code in Figure 12

5.8.2 Writing to a Page Frame

As seen in Figure 12, *KlgPfFramePrint()* can be used to write text into a template frame that will repeat from page to page. You can also use *KlgPfFramePrint()* to write to a frame on a Page currently being rendered.

For example, imagine you've created some code to recognize when a footnote is required. Unlike a fixed footer frame, you don't always want to take up white space at the bottom of a page with a footnote frame. Instead, whenever a footnote is required, you have code that generates a frame just above the footer frame:

```
footnote_frame = KlgPfFrameCreate(page, "footnote");
KlgPfFrameSetX(footnote_frame, 1.0);
KlgPfFrameSetY(footnote_frame, 9.5);
KlgPfFrameSetWidth(footnote_frame, 6.5);
KlgPfFrameSetHeight(footnote_frame, 0.5);
```

Into this frame, you use *KlgPfFramePrint()* to flow footnote text:

```
style = KlgPfDocStringToStyle(doc, "Normal");
footnote_frame = KlgPfPageStringToFrame(page, "footnote");
KlgPfFramePrint(footnote_frame, style,
                "1 - not for public disclosure");
```

5.9 Page Number Macros

A page number will be automatically generated if *KlgPfFlowPrint()* or *KlgPfFramePrint()* detects an embedded macro. Macros are embedded with an @ sign followed by one of the following number-style flags:

Flag	Function
P	The Document's `PageNumber` attribute as an Arabic page number (1, 2, 3...).
r	The Document's `PageNumber` attribute as a Roman page number, lower case (i, ii, iii...).
R	The Document's `PageNumber` attribute as a Roman page number, upper case (I, II, III...).
T	The Document's `PageCount` attribute (that is, the total number of pages in the document).
t	The total page numbers in a section (specifically, it's the last page number before resetting the Document object's `PageNumber` attribute).

5.9.1 Use @@ in Text

To render an actual @ in text, pass @@.

5.9.2 T vs. t

The *P*, *r*, and *R* flags are self-evident. The difference between *t* and *T* may not be as clear.

In our discussion of the difference between the Document object's `PageNumber` attribute and the `PageCount` attribute, it was noted that while both `PageCount` and `PageNumber` are incremented by 1 when a new page is generated, only the `PageNumber` attribute can be set by the developer with the *KlgPfDocSetPageNumber()* method. For example, you have a report with three sections, you want each section to begin again at page 1. Each time you begin a new section, your program calls *KlgPfDocSetPageNumber()* and re-sets `PageNumber` to `1`. However, you want each page of each section to display both the *current* page number and *total* number of pages in *that section.*

For example:

```
Of Recognition by Sight             Page 3 of 14
Concerning Irregular Figures        Page 9 of 22
Of the Ancient Practice of Painting Page 14 of 35
```

You can do this by including the @*t* macro.

For example:

```
KlgPfFramePrint(footer_frame, style, "Of Recognition by Sight");
KlgPfFrameTab(footer_frame, style);
```

```
KlgPfFramePrint(footer_frame, style, "Page @P of @t");

KlgPfFramePrint(footer_frame, style, "Concerning Irregular Figures");
KlgPfFrameTab(footer_frame, style);
KlgPfFramePrint(footer_frame, style, "Page @P of @t");

KlgPfFramePrint(footer_frame, style, "Of the Ancient Practice
   of Painting");
KlgPfFrameTab(footer_frame, style);
KlgPfFramePrint(footer_frame, style, "Page @P of @t");
```

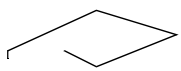
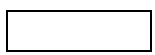
The *@t* macro will display the total number of pages rendered before *KlgPfDocSetPageNumber()* was called to re-set `PageNumber`.

Using Macros in Flowed Text

Macros are best used in non-flow frames where there is room to put the page number once it is determined. In flow code, PageFormatter cannot guarantee that it can fit the macro expansion on the current line. When PageFormatter encounters a page macro it cannot resolve until a later time, it estimates how much space must be left and continues rendering the rest of the flow until the macro can be resolved. Thus, it is best to use *KlgPfDocGetPageNumber()* or *KlgPfDocGetPageCount()* to put the current page number or count in the flow code. Even in non-flow frames, if there is not enough space, you may get overlapping text.

5.10 Drawing Tools

The Frame object provides a set of methods (defined in *KlgPfDraw.h*) for rendering circles, lines, and polygons in the frame. The table below lists the drawing methods available:

Method	Description	
KlgPfDrawSimpleLine()	Draws a line.	
KlgPfDrawPolyline()	Draws multiple connected line segments.	
KlgPfDrawFilledPolyline()	Draws multiple line segments and fills them.	
KlgPfDrawRectangle()	Draws a rectangle.	

Method	Description	
KlgPfDrawFilledRectangle()	Draws a filled rectangle.	
KlgPfDrawRoundedRectangle()	Draws a rectangle with rounded corners.	
KlgPfDrawFilledRoundedRectangle()	Draws a filled rectangle with rounded corners.	
KlgPfDrawCircle()	Draws a hollow circle.	
KlgPfDrawFilledCircle()	Draws a filled circle.	

5.10.1 DrawStyle Object

Before using a drawing method, you may want to create a DrawStyle object. The lines and fills used by the drawing method are rendered based on the DrawStyle object's attributes.

Attribute	Type
Color	KlgPfObject
FillRule	KlgPfFillRule
LineWidth	double
Name	String

For more information on the DrawStyle object, its methods, and its attributes, see Appendix D on page 163.

In the sample code below, we'll create a DrawStyle object and set its attributes.

```
KlgPfObject blackdrawstyle;
String name =  "BlackDrawStyle";
double width = 0.02;
String color = KlgPfColorCreateFromString("Black");

blackdrawstyle = KlgPfDrawStyleCreate(doc, name);
KlgPfDrawStyleSetColor(blackdrawstyle, color);
```

```
KlgPfDrawStyleSetFillRule(blackdrawstyle,
    KLGPF_FILL_RULE_NON_ZERO_WINDING);
/* sets line width */
KlgPfDrawStyleSetLineWidth(blackdrawstyle, width);
```

Creating a DrawStyle object

The DrawStyle object is created with *KlgPfDrawStyleCreate()*. A DrawStyle object is a child of the document object. *KlgPfDrawStyleCreate()* is passed a Document object and a String value, which sets the `Name` attribute. When you create a DrawStyle object, it gets added to the Document object's *DrawStyleList* list.

Color Attribute

The DrawStyle object's `Color` attribute controls a drawing method's line color and fill color. The `Color` attribute is set with *KlgPfDrawStyleSetColor()*. It is passed a DrawStyle object and Color object.

FillRule Attribute

The DrawStyle object's `FillRule` attribute is used by the *KlgPfDrawFilledPolyline()* drawing method to determine how it should fill overlapping areas. Valid values are:

KLGPF_FILL_RULE_EVEN_ODD	Defines a point to be inside if any infinite ray starting at the point crosses the polyline an odd number of times.
KLGPF_FILL_RULE_NON_ZERO_WINDING	Defines a point to be inside if any infinite ray starting at the point crosses an unequal number of clockwise or counterclockwise segments.

LineWidth

The DrawStyle object's `LineWidth` attribute controls a line's width. The `LineWidth` attribute is set with *KlgPfDrawStyleSetLineWidth()*. It is passed a DrawStyle object and double value indicating width in document units.

5.10.2 Using the Drawing Tools

KlgPfDrawSimpleLine()

KlgPfDrawSimpleLine() draws a single line between two points. Points are specified using PageFormatter's `KlgPfPoint` structure:

```
typedef struct {
   double           x, y;
} KlgPfPoint;
```

Setting CurrentDrawStyle

The `CurrentDrawStyle` attribute is set with the Document object's *KlgPfDocSetCurrentDrawStyle()* method. The default DrawStyle object creates a thin black line or a black fill and uses the `KLGPF_FILL_RULE_NON_ZERO_WINDING` rule for filling a polyline.

All drawing methods render line color, line width, and fills based on *CurrentDrawStyle*'s DrawStyle object.

For example:

```
KlgPfPoint  point1, point2;
KlgPfObject blackdrawstyle;
String name = "BlackDrawStyle";
double width = 1;
String color = KlgPfColorCreateFromString("Black");

/*  Create a DrawStyle object that will be used by
    KlgPfDrawSimpleLine */
blackdrawstyle = KlgPfDrawStyleCreate(doc, name);
KlgPfDrawStyleSetColor(blackdrawstyle, color);
KlgPfDrawStyleSetFillRule(blackdrawstyle,
    KLGPF_FILL_RULE_NON_ZERO_WINDING);
KlgPfDrawStyleSetLineWidth(blackdrawstyle, width);

/*  Set the Document's CurrentDrawStyle attribute.
    All drawing methods use this attribute */
KlgPfDocSetCurrentDrawStyle(document, blackdrawstyle);

/* Set the x, y coordinates of the KlgPfPoint structures */
point1.x = 0.2;
point1.y = 0.2;
point2.x = 0.4;
point2.y = 0.4;

/* Render the simple line */
KlgPfDrawSimpleLine(frame, point1, point2);
```

KlgPfDrawFilledPolyline()

KlgPfDrawFilledPolyline() draws multiple line segments and fills the polygon. If you do not close the polyline, *KlgPfDrawFilledPolyline()* will join the beginning and ending points.

KlgPfDrawFilledPolyline() is passed the Frame object, an array of `KlgPfPoint` structures (each `KlgPfPoint` represents a node in the polyline), and the number of points in the array.

For example:

```
/*  Create the nodes */
static KlgPfPoint ptList[4] = { {0.1, 0.1}, {1.0, 1.0}, {0.5, 1.5},
    {0.1, 0.1} };

/*  Render the polyline */
KlgPfDrawFilledPolyline(frame, ptList, 4);
```

KlgPfDrawCircle()

KlgPfDrawCircle() draws a circle. *KlgPfDrawCircle()* is passed the Frame object, a `KlgPfPoint` defining the location of the circle's center, and the circle's radius.

In the example below, we'll render a circle in the center of the frame:

```
KlgPfPoint center;
double radius = 4;

center.x = KlgPfFrameGetWidth(frame)/2;
center.y = KlgPfFrameGetHeight(frame)/2;

/*  Render the circle */
KlgPfDrawCircle(frame, &center, radius);
```

KlgPfDrawRoundedRectangle()

KlgPfDrawRoundedRectangle() draws a rectangle with rounded corners. The method is passed the Frame object, a handle to a `KlgPfRectangle` structure, and the radius of each corner arc (in document units).

A `KlgPfRectangle` structure is defined as:

```
typedef struct {
   double        x, y, width, height;
   KlgPfUnit     unit;
} KlgPfRectangle;
```

The `x` and `y` members are points relative to the Frame's top-left corner, in document units. The `width` and `height` members are the rectangle's width and height. Note, the `unit` member is ignored (document units are used).

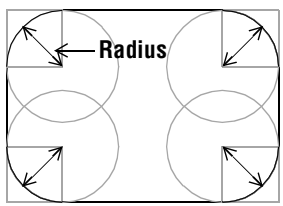
You can render a rounded rectangle in the following manner:

```
KlgPfRectangle loc;

loc.x = 0.1;
loc.y = 0.3;
loc.width = 0.8;
loc.height = 0.4;
radius = 0.2;

KlgPfDrawRoundedRectangle(frame, &loc, radius);
```

The rectangle's corner arcs are defined using the radius parameter in the following manner:



5.10.3 Ruling Lines

The Frame object's drawing tools provide a way of rendering graphics at a specific point in a frame, independent of the flow. The Document object and the Frame object each provide two methods for rendering a horizontal ruling line in the flow:

Object	Method	Description
Document Object	*KlgPfFlowHRuleLength()*	Draws a horizontal ruling line in the flow, of a specified length (to a maximum equal to column width). The line's vertical width and line color are defined by the Document's *CurrentDrawStyle* attribute.
	KlgPfFlowHRulePercent()	Draws a horizontal ruling line in the flow. The line's length is expressed as a percentage of the current column's width. The line's vertical width and line color are defined by the Document's *CurrentDrawStyle* attribute.
Frame Object	*KlgPfFrameHRuleLength()*	Draws a horizontal ruling line in the frame, of a specified length (to a maximum equal to column width). You need to pass this method a specific DrawStyle object.
	KlgPfFrameHRulePercent()	Draws a horizontal ruling line in the frame. The line's length is expressed as a percentage of the column width.You need to pass this method a specific DrawStyle object.

6

Style Object

Style Object Attributes ■ *Creating a Style Object* ■ *Copying a Style*
Setting Style Attributes ■ *Applying New Styles During Rendering*
Changing Font Appearance ■ *Changing Paragraph Appearance*

In previous examples, we have rendered text in a very plain manner, using PageFormatter's default style. PageFormatter can render paragraphs, words, and characters using many of the font and paragraph formatting features found in word processors: bold/italics, multiple fonts, centered text, and so forth.

You alter the appearance of text by creating or modifying Style objects. A Style object's attributes identify how the rendered text should appear.

6.1 Style Object Attributes

Attribute	Type
Alignment	KlgPfParagraphAlignment
Color	KlgPfObject
Font	KlgPfObject
FontFamily	KlgPfObject
FontSize	int
FontStyle	KlgPfFontStyle
IndentFirstLine	double
IndentLeft	double
IndentRight	double
LineSpacing	double
Name	String

Attribute	Type
ParagraphSpacing	double
SubscriptRatio	double
TabList	KlgPfObject
TextPosition	KlgPfTextPosition
Underline	Boolean

6.2 Creating a Style Object

When you create a Document object, PageFormatter generates a default style. For a simple document, you might only need to modify one or two attributes of the default style. For most jobs, it's easier to create several Style objects to handle various changes in font and paragraph format. For example, you can create multiple Style objects like "Heading1", "Heading2", and "BlockQuote" to take care of common paragraph styles.

When you create a Style object, it gets added to the Document object's `StyleList` list attribute. You can add several Style objects to this list for later use in your document.

To create a Style object, call the *KlgPfStyleCreate()* method. *KlgPfStyleCreate()* is passed a parent Document object and a string *name* parameter. The name attribute allows you to retrieve the style by name when needed. The method returns a Style object.

6.3 Copying a Style

The *KlgPfStyleCopy()* method can be used to create a new style object identical to an existing one.

For example:

```
heading_2 = KlgPfStyleCopy(heading_1, doc, "heading2");
```

The Style object you wish to copy is passed as the *source* parameter. You need to pass the method the new Style's parent Document object because the returned Style object is added to the Document object's `StyleList` list attribute. The *name* parameter specifies the name of the new style. It must be a unique name or else PageFormatter will simply pass back the already named style.

6.4 Setting Style Attributes

When you create a Style object, you will need to modify the object's attributes to conform to the desired text appearance. You use the Style object's provided get and set methods to modify the attributes.

For example, to change the `IndentFirstLine` attribute, which indicates how far in PageFormatter should indent the first line of a new paragraph, you would call the *KlgPfStyleSetIndentFirstLine()* method. The method is passed a Style object and an indent setting. *KlgPfStyleSetIndentFirstLine()* returns `True` if the frame's `IndentFirstLine` attribute is set successfully.

A complete list of these methods is available in Appendix N on page 243.

6.4.1 Setting Style Attributes: A Quicker Method

As you can see in the table on page 67, the Style object has a large number of properties. While you can call set methods to configure each attribute of every style you create, you may find it quicker to use *KlgPfStyleCopy()* to create a style based on an existing style and then use set methods to change only the attributes that need to differ.

For example, imagine you've created a top-level Heading1 style. You can create lower-level Heading2 and Heading3 styles by copying the Heading1 style and re-setting appropriate attributes.

```
heading_2 = KlgPfStyleCopy(heading_1, doc, "heading2");
KlgPfStyleSetFontSize(heading_2, 14);
KlgPfStyleSetIndentFirstLine(heading_2, 0.25);
KlgPfStyleSetIndentLeft(heading_2, 0.25);

heading_3 = KlgPfStyleCopy(heading_1, doc, "heading3");
KlgPfStyleSetFontSize(heading_3, 12);
KlgPfStyleSetIndentFirstLine(heading_3, 0.50);
KlgPfStyleSetIndentLeft(heading_3, 0.50);
```

6.5 Applying New Styles During Rendering

PageFormatter renders text according to the Style object pointed to by the Document object's `CurrentStyle` attribute. When you want to change the appearance of the text, you can:

- get the current style and use an appropriate set method to change the attribute;

or

- set the `CurrentStyle` attribute to a new Style object (for example, a Style object found in the Document object's `StyleList`).

6.5.1 Getting and Altering the Current Style Object

The Document object's `CurrentStyle` attribute is retrieved using the *KlgPfDocGetCurrentStyle()* method.

As an example, we will modify a portion of the flow example from Figure 5 on page 31.

```
curr_style = KlgPfDocGetCurrentStyle(doc);

KlgPfStyleSetFontStyle(curr_style, KLGPF_FONT_ITALIC);
KlgPfFlowPrint(doc, "Flatland: A Romance of Many Dimensions");
KlgPfFlowFrameFeed(doc);

KlgPfStyleSetFontStyle(curr_style, KLGPF_FONT_NORMAL);
KlgPfFlowPrint(doc, "Edwin A. Abbott");
KlgPfFlowLineFeed(doc);
KlgPfFlowPrint(doc, "(1838-1926)");
```

Figure 14 Modifying attributes of the current Style object

The program gets the current Style object with *KlgPfDocGetCurrentStyle()*. The method is passed the Document object and returns the Style object which is currently being used to format rendered text.

The *KlgPfStyleSetFontStyle()* method is called to change the current style's `FontStyle` attribute. The method is passed the Style object and an enumerated style type. For more information on font style, see Section 6.6.4 on page 74.

The text passed to *KlgPfFlowPrint()* is rendered in italic face. The program calls *KlgPfStyleSetFontStyle()* to switch the font style back to the normal style, rendering the remaining text in a regular font face.

6.5.2 Setting the CurrentStyle Attribute to a New Style Object

You should use the method shown in Figure 14 when you want to alter the appearance of a portion of text within a paragraph but not attributes of the paragraph itself. When you want to alter many aspects of the appearance of the text in the next paragraph, you should replace the `CurrentStyle` attribute with a different Style object.

Modifying the example from Figure 14, we can set our cover page text in the following manner:

```
heading_1 = KlgPfDocStringToStyle(doc, "heading1");
KlgPfDocSetCurrentStyle(doc, heading_1);
KlgPfFlowPrint(doc, "Flatland: A Romance of Many Dimensions");
KlgPfFlowFrameFeed(doc);

heading_2 = KlgPfDocStringToStyle(doc, "heading2");
KlgPfDocSetCurrentStyle(doc, heading_2);
KlgPfFlowPrint(doc, "Edwin A. Abbott");
KlgPfFlowLineFeed(doc);
KlgPfFlowPrint(doc, "(1838-1926)");
```

Figure 15 Replacing the current style with a new style

To get a style from the Document's `StyleList`, the code in Figure 15 uses the *KlgPfDocStringToStyle()* method. The method is passed the Document object and the `Name` attribute of the Style object. It returns the named Style object from the Document object's `StyleList`.

The program passes the *heading1* Style object to the *KlgPfDocSetCurrentStyle()* method to set the Document object's `CurrentStyle` attribute. This method gets passed a Document object and a Style object. It returns `TRUE` if the `CurrentStyle` attribute is set. The text rendered with *KlgPfFlowPrint()* using the *heading1* style attributes.

A few lines down, the program repeats the process, this time getting the *heading2* style from the Document object's `StyleList` and then passing it to the *KlgPfDocSetCurrentStyle()* method. The text now gets rendered using the *heading2* style attributes.

6.6 Changing Font Appearance

Many documents require frequent changes in the appearance of the font: A group of words will need to be bolded, a character will need to be rendered as a superscript, or a heading will need to be rendered using a different font family (for example, your body text is "Times" but your heading text is "Helvetica").

The Style object provides seven attributes to control the appearance of fonts:

Attribute	Type
Font	KlgPfObject
FontFamily	KlgPfObject
FontSize	int
SubscriptRatio	double
FontStyle	KlgPfFontStyle
TextPosition	KlgPfTextPosition
Underline	Boolean

Modifying the example from Figure 15, we change the font family and font style of our cover page text in the manner displayed in Figure 16.

```
curr_style = KlgPfDocGetCurrentStyle(doc);
fontfamily = KlgPfPrinterStringToFontFamily(printer, "Helvetica");

KlgPfStyleSetFontFamily(curr_style, fontfamily);
KlgPfStyleSetFontStyle(curr_style, KLGPF_FONT_ITALIC);
KlgPfFlowPrint(doc, "Flatland: A Romance of Many Dimensions");
KlgPfFlowFrameFeed(doc);

fontfamily = KlgPfPrinterStringToFontFamily(printer, "Times")

KlgPfStyleSetFontFamily(curr_style,fontfamily);
KlgPfStyleSetFontStyle(curr_style, KLGPF_FONT_NORMAL);
KlgPfFlowPrint(doc, "Edwin A. Abbott");
KlgPfFlowLineFeed(doc);
KlgPfStyleSetFontSize(curr_style, 9);
KlgPfFlowPrint(doc, "(1838-1926)");
```

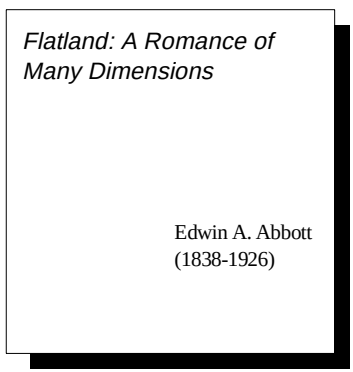
Figure 16 Changing Font Style and Font Family

6.6.1 Font Family

In Figure 16, the *KlgPfPrinterStringToFontFamily()* method is used to retrieve the Helvetica font family from the Printer object's `FontFamilyList` list. The *KlgPfPrinterStringToFontFamily()* method is passed a Printer object and the string `Name` attribute of the FontFamily object. It returns the FontFamily object. (The `FontFamilyList` list attribute is not part of the Document or Style object as different printers will have different fonts, hence different font families.)

The current Style object's `FontFamily` attribute is set with the *KlgPfStyleSetFontFamily()* method. The method is passed a Style object and a FontFamily object. It returns `True` if the `FontFamily` attribute is set.

The code from Figure 16 would produce the following output:



6.6.2 Font Family Vs. Font

The `FontFamily` attribute is an object that provides methods and a structure for grouping together individual but related fonts into a "font family" (for example

"Times"). Technically, there is no "Times" font. As Figure 17 demonstrates, "Times" is a label to describe a family of fonts encompassing "Times-Roman", "Times-Bold", "Times-Italic", and "Times-BoldItalic". All are separate fonts as far as a printer and the rendering process is concerned.

The `FontFamily` attribute in conjunction with the `FontStyle` attribute provides an interface that allows you to select a font family and apply a font style (like bold) to that font.

```
Times-Roman            Helvetica               NewCenturySchlbk-Roman
Times-Bold             Helvetica-Bold          NewCenturySchlbk-Bold
Times-Italic           Helvetica-Oblique       NewCenturySchlbk-Italic
Times-BoldItalic       Helvetica-BoldOblique   NewCenturySchlbk-BoldItalic

Palatino-Roman         Bookman-Light           Helvetica-Narrow
Palatino-Bold          Bookman-Demi            Helvetica-Narrow-Bold
Palatino-Italic        Bookman-LightItalic     Helvetica-Narrow-Oblique
Palatino-BoldItalic    Bookman-DemiItalic      Helvetica-Narrow-BoldOblique

Courier                AvantGarde-Book         Symbol
Courier-Bold           AvantGarde-Demi         ZapfChancery-MediumItalic
Courier-Oblique        AvantGarde-BookOblique  ZapfDingbats
Courier-BoldOblique    AvantGarde-DemiOblique
```

Figure 17 The 35 Standard LaserWriter II Font Names

6.6.3 Font and FontFamily Lists

The Printer object maintains a list of supported fonts in its `FontList` attribute. It maintains a list of font families in its `FontFamily` attribute. X11 printers get a list of fonts from the server. When you create a PostScript printer, this list is generated automatically based on supplied AFM files in *$KLGPRHOME/lib/ps/afm*. Fonts are added to the `FontList` attribute as Font objects. Font families are added to the `FontFamily` attribute as Font Family objects.

If you add new fonts, you must put the AFM file in *$KLGPRHOME/lib/ps/afm*.

To retrieve a list of specific fonts and font families, you can:

- Use the Printer object's *KlgPfPrinterGetFontFamilyList()* and *KlgPfPrinterGetFontList()* methods to return the desired list object.
- Use the List object's *KlgPfListGetItemCount()* method to determine the number of items on the list.
- Call *KlgPfListGetItem()* to retrieve individual list objects by index position.
- Use the Font object's *KlgPfFontGetName()* method and the Font Family object's *KlgPfFontFamilyGetName()* to retrieve the font and font family `Name` attributes.

6.6.4 Font Style

In Figure 16, the current Style object's `FontStyle` attribute is changed by passing the current Style object along with style values to the *KlgPfStyleSetFontStyle()* method. Valid values are:

`KLGPF_FONT_NORMAL`	Renders the text with no bold or italics.
`KLGPF_FONT_BOLD`	Renders the text in bold.
`KLGPF_FONT_ITALIC`	Renders the text in italics.
`KLGPF_FONT_BOLDITALIC`	Renders the text in bold and italic.

6.6.5 Font Size

In Figure 16, we want to retain the font style and font family but decrease the font size to 9 points. Font size is specified by the Style object's `FontSize` attribute. The `FontSize` attribute is set using the *KlgPfStyleSetFontSize()* method. The method is passed a Style object and an integer indicating the font size in points. It returns `True` if the `FontSize` attribute is set.

6.6.6 Subscripts/Superscripts

Two Style attributes control the rendering of subscripts and superscripts. The `SubscriptRatio` attribute specifies the size ratio of the superscript/subscript font to the "regular" font size as defined by the `FontSize` attribute. For example, a 12 point `FontSize` and a 0.75 `SubscriptRatio` will produce a 9 point superscript/subscript.

If you need to adjust this attribute, you can do so using the *KlgPfStyleSetSubscriptRatio()* method.

You can get PageFormatter to render subscript/superscript text by setting the `TextPosition` attribute with the *KlgPfStyleSetTextPosition()* method. The method is passed a Style object and a position value. Valid values are:

`KLGPF_FONT_ON_BASELINE`	Renders text in the normal fashion, along the baseline.
`KLGPF_FONT_SUBSCRIPT`	Renders text below the baseline based on the `SubscriptRatio` attribute.
`KLGPF_FONT_SUPERSCRIPT`	Renders text above the baseline based on the `SubscriptRatio` attribute.

It returns `True` if the `TextPosition` attribute is set.

For example to produce "This is a footnote[1]. This is not.", you could write the following:

```
curr_style = KlgPfDocGetCurrentStyle(doc);
KlgPfFlowPrint(doc, "This is a footnote");
KlgPfStyleSetTextPosition(curr_style, KLGPF_FONT_SUPERSCRIPT);
KlgPfFlowPrint(doc, "1");
KlgPfStyleSetTextPosition(curr_style, KLGPF_FONT_ON_BASELINE);
KlgPfFlowPrint(doc, ". This is not.");
```

The program sets `TextPosition` to render text that follows as a superscript. The 1 footnote character is flowed. The program then sets the `TextPosition` attribute back to its baseline rendering position and the remaining text is flowed as usual.

6.6.7 Underlining

Underling is specified by the `Underline` attribute. It is a simple Boolean attribute indicating whether to render text with an underscore or not. Underlining can be enabled or disabled with the *KlgPfStyleSetUnderline()* method. The method is passed a Style object and is either `True` (which enables underlining) or `False` (which disables underlining).

6.7 Changing Paragraph Appearance

While font-related attributes can be adjusted for individual characters or words, Style object attributes that specify indents, alignment, line spacing, and paragraph spacing should be set in relation to an entire paragraph.

The Style object provides six attributes to control the appearance of whole paragraphs:

Attribute	Type
Alignment	KlgPfParagraphAlignment
IndentFirstLine	double
IndentRight	double
IndentLeft	double
LineSpacing	double
ParagraphSpacing	double

6.7.1 Alignment

The `Alignment` attribute specifies paragraph justification. It is set with the *KlgPfStyleSetAlignment()* method.

For example:

```
KlgPfStyleSetAlignment(curr_style, KLGPF_ALIGN_CENTER);
```

The method is passed a Style object and an alignment value. Valid values are:

KLGPF_ALIGN_LEFT	Paragraph text is left aligned.
KLGPF_ALIGN_CENTER	Paragraph text is centered.
KLGPF_ALIGN_RIGHT	Paragraph text is right aligned.
KLGPF_ALIGN_JUSTIFY	Paragraph text is left and right justified.

Note: you cannot use KLGPF_ALIGN_CENTER, KLGPF_ALIGN_RIGHT, or KLGPF_ALIGN_JUSTIFY alignments with tabs. If *KlgPfFlowTab()* is called while the alignment is set to a value other than KLGPF_ALIGN_LEFT, PageFormatter will set the Alignment attribute to KLGPF_USING_TABS.

6.7.2 Indents

Three indent attributes control how far the text should be rendered from the edge of the frame.

First Line Indent

The IndentFirstLine attribute specifies, in document units, how far the text in the *first* line of a paragraph is rendered from the left side of the frame or column. It is set with the *KlgPfStyleSetIndentFirstLine()* method.

Left Indent

The IndentLeft attribute specifies, in document units, how far lines of text subsequent to the first line is rendered from the left side of the frame or column. This attribute affects all lines in the paragraph, except the first line. It is set with the *KlgPfStyleSetIndentLeft()* method.

Right Indent

The IndentRight attribute specifies, in document units, how far text is rendered from the right side of the frame or column. This attribute affects all lines in the paragraph, including the first line. It is set with the *KlgPfStyleSetIndentRight()* method.

6.7.3 Line Spacing

The LineSpacing attribute specifies the amount, in document units, paragraph lines are spaced apart vertically. It is set with the *KlgPfStyleSetLineSpacing()* method.

For example:

```
KlgPfStyleSetLineSpacing(curr_style, 0.19);
```

The method is passed a Style object and a line spacing amount. Assuming text font size is 12 points and the document unit is “Inches”, 0.19" double-line spacing would be produced.

For most methods, passing values in terms default document unit of inches or centimeters is most convenient. However, for finer control, like line spacing and paragraph spacing, it's easier to work in points. To aid you, PageFormatter provides a number of conversion functions (see Appendix Q on page 297 for a complete list of functions).

Using the *KlgPfConvertPointsToInches()* method, the example above could be re-written:

```
KlgPfStyleSetLineSpacing(curr_style, KlgPfConvertPointsToInches(14));
```

6.7.4 Paragraph Spacing

The `ParagraphSpacing` attribute specifies the amount of white space that should follow a completely rendered paragraph. Paragraph spacing (or "leading" as it is known in the field of typography) is used to help visually separate paragraphs. The `ParagraphSpacing` attribute is set with the *KlgPfStyleSetParagraphSpacing()* method.

6.7.5 Tab List

The Style object also maintains a list of Tab objects with the `TabList` list attribute. Each Tab object on the list specifies a tab stop. For more information on tabs, see "Tab Object" on page 79.

7

Advanced Formatting

Tab Object ■ *Color Object* ■ *Images*

7.1 Tab Object

The Tab object is a child of the Style object. The Tab object provides methods for adding tabs to the Style object's `TabList` attribute and modifying the tab's type, horizontal position, and appearance.

7.1.1 Tab Object Attributes

Attribute	Type
FillType	KlgPfFillType
TabType	KlgPfTabType
X	double

7.1.2 Creating a Tab Object

By default, PageFormatter creates a left-aligned tab stop every half inch. You may need to create different tabs to address different formatting needs. A Tab object is created with the *KlgPfTabCreate()* method.

For example:

```
style = KlgPfDocStringToStyle(doc, "heading1");
new_tab = KlgPfTabCreate(style, 5.2);
```

The method is passed a Style object and a value indicating the Tab object's tab stop relative to the frame or column. When you create a Tab object, it gets added to the indicated Style object's `TabList`. `TabList` is maintained in sorted order. The method returns the Tab object.

By default, this will create a left-aligned tab with no fill attribute.

7.1.3 Setting Fill Type

It is sometimes preferable to fill in the space bridged by a tab with a "leader": a row of periods (.....) or underscores (___). If large gaps exist between columns of text, the leader helps the eye track horizontally from item to item.

The Tab object's `FillType` attribute is used to specify the leader type. It is set with the *KlgPfTabSetFillType()* method.

The method is passed a Tab object and a fill type. Valid values are:

`KLGPF_TAB_NO_FILL`	No fill. Blank space.
`KLGPF_TAB_FILL_WITH_UNDERSCORE`	Fill with underscore ("_") characters.
`KLGPF_TAB_FILL_WITH_ELLIPSES`	Fill with period ("...") characters.

7.1.4 Setting the Tab Type

PageFormatter supports several common tab types:

Left-aligned tab	Characters following the tab are aligned on the left, at the tab's X value
Right-aligned tab	Characters following the tab are aligned on the right, at the tab's X value
Center tab	Characters following the tab are aligned centered at the tab's X value
Decimal tab	Characters following the tab are aligned by their decimal place at the point defined by the tab's X value.

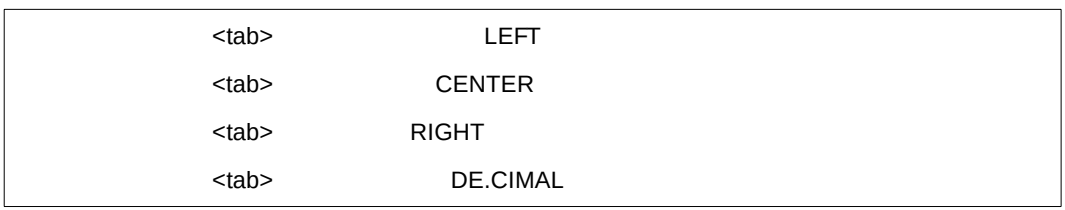


Figure 18 Different tab styles set at the 3" mark.

The Tab object's `TabType` attribute is used to specify the tab type. It is set using the *KlgPfTabSetTabType()* method.

The method is passed a Tab object and a tab type. Valid values are:

`KLGPF_TAB_RIGHT`	Right aligned tab
`KLGPF_TAB_LEFT`	Left aligned tab

KLGPF_TAB_CENTER	Center tab
KLGPF_TAB_DECIMAL	Decimal tab

7.1.5 Tab Flow Method

To actually use a Tab object, you need to call the *KlgPfFlowTab()* method.

For example, to render "1.	I'm following a left aligned tab.", you could use the following:

```
KlgPfFlowPrint(doc, "1.");
KlgPfFlowTab(doc);
KlgPfFlowPrint(doc, "I'm following a left aligned tab.");
```

The method is passed the Document object. It returns `True` if the tab feed is successful.

KlgPfFlowTab() generates a tab feed in the current flow. The method reads the `TabList` list attribute of the Style object pointed to by the specified the Document object's `CurrentStyle` attribute. As well, it determines the current X position relative to the frame or column. The method finds the *next* tab object on the `TabList` with an X position *greater* than the current X position and renders according to the attributes of that tab object. If there is no tab in the `TabList` that is greater than the current X, PageFormatter flows to the next line.

7.1.6 Setting Multiple Left-Aligned Tabs

While you can set single tabs, PageFormatter provides two convenience functions to set multiple left-aligned tabs.

Setting Equally Spaced Tabs

KlgPfStyleSetTabsEquallySpaced() lets you set a number of equally spaced tabs from a given X position. Suppose we want to set six tabs, spaced half an inch apart, beginning at the 1" mark. *KlgPfStyleSetTabsEquallySpaced()* would be passed the following:

```
KlgPfStyleSetTabsEquallySpaced(style, 1, 0.5, 6);
```

The method is passed the Style object to which you want to add the tabs, the start value (in this case the 1" mark), the increment value (0.5"), and the number of tabs. In this example, tabs at 1.5", 2.0", 2.5", 3.0", 3.5", 4.0" would be created.

Setting Tabs from an Array

KlgPfStyleSetTabsByValueList() is passed an array of X values and will create left-aligned tabs at the X positions denoted by the array. For example,

```
double tabvalues[] = {1.5, 2.0, 2.5, 3.0, 3.5, 4.0}
KlgPfStyleSetTabsByValueList(style, tabvalues, 6)
```

The method is passed the Style object to which you want to add the tabs, the array, and an integer value indicating the number values in the array. PageFormatter assumes the accuracy of this parameter.

7.1.7 Clearing Tabs

You can clear all tabs on the `TabList` with the *KlgPfStyleClearTabs()* method.

7.1.8 Creating a Bullet with Tabs and Indent

You can create a bullet by setting the Style object's `IndentLeft` setting greater than the Style object's `FirstLine` indent and a tab equal to `IndentLeft`.

For example:

```
style = KlgPfStyleCreate(doc, "bullet");
KlgPfStyleSetIndentFirstLine(style, 0);
KlgPfStyleSetIndentLeft(style, 0.25);
KlgPfTabCreate(style, 0.25);

KlgPfDocSetCurrentStyle(doc, style);
KlgPfFlowPrint(doc, "*");
KlgPfFlowTab(doc);
KlgPfFlowPrint(doc, "KlgPfFlowTab() advances this text to the tab
   position. When the text flows to the next line, the
   IndentLeft setting moves all subsequent lines flush with
   the tab.");
```

The above code would produce:

* KlgPfFlowTab() advances this text to the tab position. When the text flows to the next line, the IndentLeft setting moves all subsequent lines flush with the tab.

7.2 Color Object

The Color object is a child of the Page, Frame, and Style object. The Color object provides methods for the creation of colors. The Color object creates a color by specifying the intensity of Red, Blue, and Green attributes. The Color object provides several named, predefined hues and lets you set one using a name string (for example "Purple"). If you're developing for a monochrome environment, the Color object also provides methods for setting grey levels.

7.2.1 Color Object Attributes

Attribute	Type
Blue	double
Green	double
Red	double

7.2.2 Creating a Color Object

There are two ways to create a Color object:

- Create the basic object and then call set methods to set its Red, Green, and Blue attributes.
- Create a Color object using a predefined color string.

A Color object is created with the *KlgPfColorCreate()* method.

For example:

```
new_color = KlgPfColorCreate();
```

No Parent object is passed to the Color object as it can be a child of a number of objects. The method returns a Color object with its `Red`, `Green`, and `Blue` attributes set to 0 (black).

Destroying a Color Object

Since the Color object has no parent object, it will not be destroyed when you destroy a Document object. You must destroy all Color objects you create with *KlgPfColorDestroy()*.

7.2.3 KlgPfColorSetRGB()

After creating the Color object, you can use the *KlgPfColorSetRGB()* method to set the object's Blue, Green, and Red attributes.

For example:

```
KlgPfColorSetRGB(new_color, 0.3, 0.2, 0.8);
```

The method is passed a Color object. The *red*, *green*, and *blue* parameters take a value between 0 and 1. 1 represents the color at its maximum intensity. 0 represents the absence of the color from the RGB mix.

7.2.4 Setting Color Attributes Individually

Alternatively, you can use the provided *KlgPfColorSetBlue()*, *KlgPfColorSetRed()*, and *KlgPfColorSetGreen()* set methods to set the `Red`, `Green`, and `Blue` attributes individually.

Each method is passed a Color object and a value between 0 and 1. 1 represent the attribute at its maximum intensity. 0 represents the absence of the color.

7.2.5 Predefined Color Strings

PageFormatter has predefined a number of RGB color settings. A complete list of these color settings (for example, "Purple" or "Gray72" can be found in Appendix A).

You can create a Color object and have its `red`, `green`, and `blue` attributes set automatically by passing one of the named colors to the *KlgPfColorCreateFromString()* method.

For example:

```
new_color = KlgPfColorCreateFromString("Purple");
KlgPfFrameSetBorderColor(body_frame, new_color);
```

The *KlgPfColorCreateFromString()* method is passed a valid color name. It returns the Color object.

7.2.6 Monochrome Environments

In a monochrome environment, you can set grey levels by setting all three color attributes to the same intensity. The *KlgPfColorCreateGrey()* method allows you to create a color object and set all three color attributes by passing only one value.

For example:

```
new_grey_level = KlgPfColorCreateGrey(0.5);
```

The method creates a Color object and sets `Red`, `Green`, and `Blue` attributes to the specified value. *KlgPfColorCreateGrey()* returns a Color object.

If you need to alter the grey level of an existing color object, you can use the *KlgPfColorSetGrey()* method the reset all three color attributes by passing only one value.

For example:

```
KlgPfColorSetGrey(new_grey_level, 0.25);
```

The method is passed an existing Color object and an intensity level between 0 and 1.

7.3 Images

Images are rendered by first creating an Image object for each image. Each supported image (EPS, XPM, GIF, JPEG) is created with its own method.

Method	Creates
KlgPfImageCreateFromEPSF()	Create an EPS Image object
KlgPfImageCreateFromGIF()	Create a GIF Image object
KlgPfImageCreateFromJPEG()	Create a JPEG Image object

Method	Creates
KlgPfImageCreateFromXPM()	Create an XPM Image object

For more information on the Image object see the Image Object on page 195.

Notes:

- Image object dimensions don't have to match the document dimensions, so they must be specified in the call to *KlgPfImageSetSize()*.
- Images can be held in memory and used in multiple documents.

7.3.1 Image Format

PageFormatter supports images in EPS, XPM (XPixMap), GIF, and JPEG formats. Images in other formats (such as TIFF or Sun Rasterfile) should be converted to one of these formats. You can use, for example, *pbmplus* (available from multiple sources on the Internet) to convert images to PBM format (a standard bitmap file format) and then convert them to XPM format.

XPM is an ASCII image format that lets you store an image in a file or in C code. It is a standard X Window format and can be easily used with all supported printers. You will not need to install the X libraries to use XPM format with PageFormatter. You may need to link in the library *libnoX11.a* (that is, add a `-LnoX11` to the link line in your makefile) to resolve some unused symbols by the XPM library.

Note: An EPS image will be displayed by default as a grey box when rendered using an X11, Acrobat, or PCL printer, since none of these have built-in support for EPS. The grey delimiting box can be turned off by setting the Image's `VisibleBoundingBox` attribute to `False`.

7.3.2 Flowing Images

PageFormatter can render images in two fashions:

- As an embedded or "inline" graphic that is part of a line of text.
- As a floating image, a graphic on its own line.

7.3.3 Embedded Graphics

Graphics can be rendered in a line of text by calling the *KlgPfFlowImageEmbed()* method.

For example:

```
1.  #include <KlgPfGIF.h>;

2.  static KlgPfRectangle size = {0, 0, 1, 1, KGLPF_UNIT_CM};
3.  FILE * input_file;
4.  KlgPfObject gif_image;
5.  input_file = fopen ("pict.gif", "r");

6.  gif_image = KlgPfImageCreateFromGIF(input_file);
7.  KlgPfImageSetSize(gif_image, &size);
8.  fclose (input_file);

9.  KlgPfFlowPrint(doc, "After I had concluded my defence, the ");
10. KlgPfFlowPrint(doc, "President, perhaps perceiving that some of ");
11. KlgPfFlowPrint(doc, "the junior Circles had been moved by my ");
12. KlgPfFlowPrint(doc, "evident earnestness, asked me two questions:");
13. KlgPfFlowLineFeed(doc);

14. KlgPfFlowImageEmbed(doc,gif_image, KLGPF_IMAGE_BOTTOM);

15. KlgPfFlowPrint(doc, " Whether I could indicate the direction which I ");
16. KlgPfFlowPrint(doc, "meant when I used the words Upward, not Northward?");
17. KlgPfFlowLineFeed(doc);

18. KlgPfFlowImageEmbed(doc,gif_image, KLGPF_IMAGE_BOTTOM);

19. KlgPfFlowPrint(doc, "Whether I could by any diagrams or descriptions ");
20. KlgPfFlowPrint(doc, "indicate the Figure I was pleased to call");
21. KlgPfFlowPrint(doc, "a Cube?");

22. KlgPfImageDestroy(gif_image);
```

Figure 19 KlgPfFlowImageEmbed() method to include an embedded graphic

On line 6, a GIF Image object is created by passing the *KlgPfImageCreateFromGIF()* method a handle to an open input file stream.

On lines 14 and 18, the image is rendered by passing the *KlgPfFlowImageEmbed()* method a Document object, an Image object, and a vertical alignment type.

Valid alignment types are:

KLGPF_IMAGE_BASELINE	Aligns the image at the text's baseline.
KLGPF_IMAGE_BOTTOM	Aligns the image at the bottom of the text's descender.
KLGPF_IMAGE_CENTER	Vertically centers the image in the line.
KLGPF_IMAGE_TOP	Aligns the image at the top of the line.

Assuming the code in Figure 19 was part of compilable code, and the image was a fancy bullet graphic, you would get the following output:

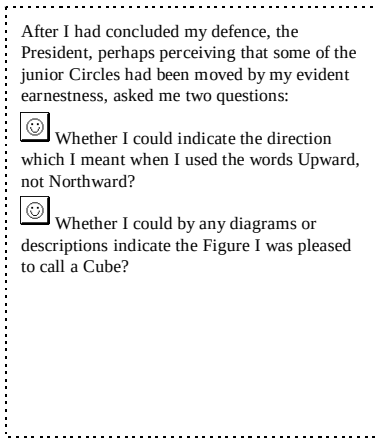


Figure 20 Result of flowing embedded images

Not Enough Space

If PageFormatter calculates there is no space left on the line to render the image in full, it will wrap the line and render the image on the next line. You should not try to embed images wider than a line in the current column.

7.3.4 Floating Images

Larger graphics can be rendered on their own line as a floating image with *KlgPfFlowImageFloat()*.

For example:

```
1   KlgPfFlowPrint(doc, "Strong economic growth and accelerating ");
2   KlgPfFlowPrint(doc, "wages prompted the Federal Fiscal Bank to ");
3   KlgPfFlowPrint(doc, "raise interest rates in late June, the first ");
4   KlgPfFlowPrint(doc, "such move in six years. Blue-chip stocks, ");
5   KlgPfFlowPrint(doc, "reported gains as interest rates rose.");
6   KlgPfFlowLineFeed(doc);
7
8   KlgPfFlowImageFloat(doc, gif_image);
```

Figure 21 KlgPfFlowFloatImage() renders an embedded graphic on its own line

On line 8, the *KlgPfFlowImageFloat()* method is passed the Document object and the Image object.

Assuming the code in Figure 21 was part of compilable code, and the image was a graph, you would get the following output:

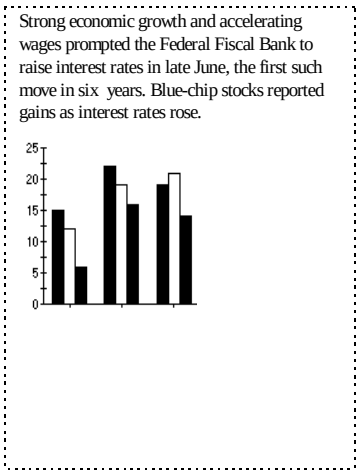


Figure 22 Result of flowing a floating image

Not Enough Space

If PageFormatter calculates there is no room left in the frame to render the image in full, it will continue rendering text in the flow after the *KlgPfFlowImageFloat()* call until a new frame or page is encountered with sufficient space. PageFormatter will then render the image in the new frame. If sufficient space is not found before *KlgPfDocEnd()*, the image will not be rendered.

8

Tables

Table Object ■ *Adding Data to the Table* ■ *Adding a Header*
Changing the Table Appearance ■ *Borders* ■ *Table Inheritance*
Setting Row, Column, and Cell Attributes ■ *Creating a Cell Span*

8.1 Table Object

The Table object is a child of a Document or Frame object. The Table object provides methods and attributes for creating tables. Using the Table object's sub-objects (the Row, Column, Cell, and Border objects) you can customize the table's rows, columns, cells, and borders.

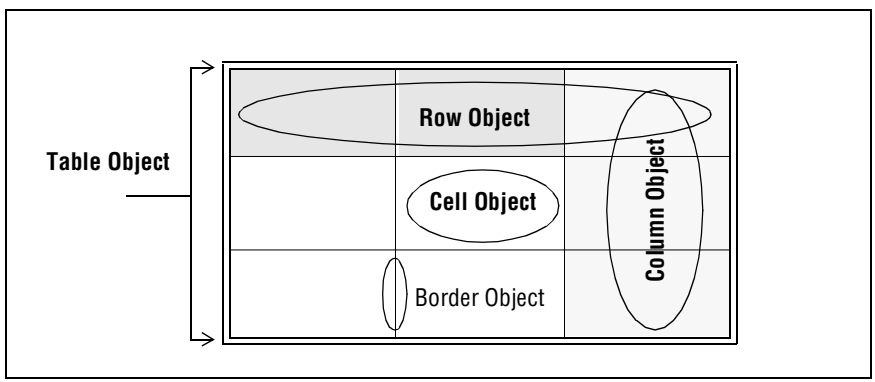


Figure 23 Objects that Comprise a Table Object

8.1.1 Table Object Attributes

Attribute	Type
Alignment	KlgPfParagraphAlignment
BackgroundColor	KlgPfObject
BorderBottom	KlgPfObject
BorderColumn	KlgPfObject
BorderHeader	KlgPfObject
BorderLeft	KlgPfObject
BorderMode	KlgPfTableBorderMode
BorderRight	KlgPfObject
BorderRow	KlgPfObject
BorderTop	KlgPfObject
CellAlignment	KlgPfCellAlignment
CellMarginBottom	double
CellMarginLeft	double
CellMarginRight	double
CellMarginTop	double
CellStyle	KlgPfObject
ColumnList	KlgPfObject
DirectionPriority	KlgPfTableDirectionPriority
HeaderTable	KlgPfObject
OverflowMode	KlgPfTableOverflowMode
RowList	KlgPfObject

8.1.2 Column Object Attributes

Attribute	Type
Alignment*	KlgPfCellAlignment
BackgroundColor*	KlgPfObject
BorderRight*	KlgPfObject
CellMarginBottom*	double
CellMarginLeft*	double
CellMarginRight*	double

Attribute	Type
CellMarginTop*	double
CellStyle*	KlgPfObject
Width	double

Note: a * indicates the object may inherit this attribute setting from its parent object. For more information on inheritance, see "Table Inheritance" on page 104.

8.1.3 Row Object Attributes

Attribute	Type
Alignment*	KlgPfCellAlignment
BackgroundColor*	KlgPfObject
BorderBottom*	KlgPfObject
CellList	KlgPfObject
CellMarginBottom*	double
CellMarginLeft*	double
CellMarginRight*	double
CellMarginTop*	double
CellStyle*	KlgPfObject

Note: a * indicates the object may inherit this attribute setting from its parent object. For more information on inheritance, see "Table Inheritance" on page 104.

8.1.4 Cell Object Attributes

Attribute	Type
Alignment*	KlgPfCellAlignment
BackgroundColor*	KlgPfObject
BorderBottom*	KlgPfObject
BorderRight*	KlgPfObject
Frame	KlgPfObject
Height	double
MarginBottom*	double
MarginLeft*	double
MarginRight*	double

Attribute	Type
MarginTop*	double
SpanOrigin	KlgPfPoint
SpanRows	int
Spanned	Boolean
Style*	KlgPfObject

Note: a * indicates the object may inherit this attribute setting from its parent object. For more information on inheritance, see "Table Inheritance" on page 104.

8.1.5 Border Object Attributes

Attribute	Type
Color	KlgPfObject
RuleType	KlgPfTableRuleType
Spacing	double
Width	double

8.1.6 Creating a Table Object

The sample code in Figure 24 creates a table object with one row and three columns. It renders it into the current flow.

```
1.    KlgPfObject table;
2.    static String celltext[] = {"Row 0 Col 0", "Row 0 Col 1", "Row 0 Col 2"};
3.    static double columns[] = {1.00, 1.25, 2.25, -1.0};
4.    int ncolumns = 3;

5.    /* print some text in the flow */
6.    KlgPfFlowPrint(doc, "Simple table example:  this program demonstrates
        construction of a simple table using PageFormatter.");

7.    /* create a table with three columns */
8.    table = KlgPfTableCreate(doc, columns, ncolumns);

9.    /* add celltext as a row */
10.   KlgPfTableAddRow(table, celltext);

11.   /* render the table in the flow and destroy the table object */
12.   KlgPfTablePrintAndDestroy(doc, table);
```

Figure 24 Flowing a simple table

On line 8, the code creates the Table object. It's a child of the Document object and has three columns. No actual rows or cells are created by *KlgPfTableCreate()*. *KlgPfTableAddRow()* on line 10 creates the first row and three cells. Each member of the `celltext[]` array is placed in an individual cell. Once the table is populated, it is rendered as part of the current document flow with *KlgPfTablePrintAndDestroy()* on line 11. The *KlgPfTablePrintAndDestroy()* method also destroys the Table object and all its child objects.

If the above code was added to a PageFormatter program, you would get the following output:

Row 0 Col 0	Row 0 Col 1	Row 0 Col 2

8.2 Adding Data to the Table

In the example in Figure 24, a table row is created and populated by the *KlgPfTableAddRow()* method. It is not the only method for adding data to the table. The Table object provides a number of methods for adding cell data to the table.

8.2.1 KlgPfTableAddRow()

The *KlgPfTableAddRow()* method adds a row to the end of the table. As demonstrated in Figure 24, *KlgPfTableAddRow()* is passed a string array. The array must contain as many entries as there are columns in the table. If you would like a cell to be blank, include `NULL` in the array.

8.2.2 KlgPfTableAddRange()

If you would like to populate a sequence of cells in a row, you can use the *KlgPfTableAddRange()* method. This method adds a series of cells. You must specify the row, initial column, cell data, and number of cells you wish to add. Similar to *KlgPfTableAddRow()* , if you would like a cell to be blank, include `NULL` in the String array.

For example:

```
int row = 0;
int column = 1;
static String celltext[] = {"Row 0 Col 1", "Row 0 Col 2"};
int cell_count = 2;

KlgPfTableAddRange(table, row, column, celltext, 2);
```

8.2.3 Printing Directly to a Cell

Cells and rows can be generated with *KlgPfTablePrintToCell()*, which allows you to print directly to cells. If the cell does not exist, PageFormatter creates the cell. If the row does not exist, *KlgPfTablePrintToCell()* will create the row. *KlgPfTablePrintToCell()*

is passed the row index of the desired cell, the column index of the desired cell, and the string you wish to render.

```
1.    KlgPfObject table;
2.    static double columns[] = {1.00, 1.25, 2.25, -1.0};
3.    int ncolumns = 3;

4.    /* print some text in the flow */
5.    KlgPfFlowPrint(doc, "Simple table example:  this program demonstrates
          construction of a simple table using PageFormatter.");

6.    /* create a table with three columns */
7.    table = KlgPfTableCreate(doc, columns, ncolumns);

8.    /* add cells and rows with KlgPfTablePrintToCell() */
9.    KlgPfTablePrintToCell(table, 0, 0, "Row 0 Col 0");
10.   KlgPfTablePrintToCell(table, 0, 1, "Row 0 Col 1");
11.   KlgPfTablePrintToCell(table, 0, 2, "Row 0 Col 2");
12.   KlgPfTablePrintToCell(table, 1, 0, "Row 1 Col 0");
13.   KlgPfTablePrintToCell(table, 1, 1, "Row 1 Col 1");
14.   KlgPfTablePrintToCell(table, 1, 2, "Row 1 Col 2");
15.   KlgPfTablePrintToCell(table, 2, 0, "Row 2 Col 0");
16.   KlgPfTablePrintToCell(table, 2, 1, "Row 2 Col 1");
17.   KlgPfTablePrintToCell(table, 2, 2, "Row 2 Col 2");

18.   /* render the table in the flow and destroy the table object */
19.   KlgPfTablePrintAndDestroy(doc, table);
```

Figure 25 Flowing a simple table with KlgPfTablePrintToCell()

If the above code was added to a PageFormatter program, you would get the following output:

Row 0 Col 0	Row 0 Col 1	Row 0 Col 2
Row 1 Col 0	Row 1 Col 1	Row 1 Col 2
Row 2 Col 0	Row 2 Col 1	Row 2 Col 2

8.2.4 Creating a Simple Table

You can create, populate, render, and destroy an entire table with the *KlgPfTableCreateSimple()* method. *KlgPfTableCreateSimple()* is passed the document or current frame you wish to render the table in, an array of column widths, the number of columns in the table, and a 2D array of Strings to be used as cell contents. A `NULL`

row terminates the 2D array. Each row of the array must have as many entries as there are columns in the table. `NULL` indicates a blank cell.

```
#define NROWS    3
#define NCOLUMNS 3

 int i;
 static double columns[] = {1.00, 1.25, 2.25, -1.0};
 static String rows[][NCOLUMNS] = {{ "Row 0 Col 0", "Row 0 Col 1", "Row 0 Col 2" },
                                   { "Row 1 Col 0", "Row 1 Col 1", "Row 1 Col 2" },
                                  { "Row 2 Col 0", "Row 2 Col 1", "Row 2 Col 2" } };
 String *cell_data[NROWS + 1];
 KlgPfObject table, current_frame;

 /* create a NULL-terminated list of rows */
  for (i = 0; i < NROWS; i++) {
  cell_data[i] = rows[i];
 }
 cell_data[NROWS] = NULL;

 /* Create the page templates and start the document */
 KlgPfPageTemplateCreate(doc, KlgPfIOBuffer, template_dat);
 KlgPfDocBegin(doc);

 KlgPfFlowPrint(doc, "Simple table example:  this program ");
 KlgPfFlowPrint(doc, "demonstrates construction of a simple table ");
 KlgPfFlowPrint(doc, "using PageFormatter.");

 current_frame = KlgPfDocGetCurrentFrame(doc);

 KlgPfTableCreateSimple(current_frame, columns, NCOLUMNS, cell_data);
```

Figure 26 Flowing a simple table

If the code was added to a PageFormatter program, you would get the following output:

Row 0 Col 0	Row 0 Col 1	Row 0 Col 2
Row 1 Col 0	Row 1 Col 1	Row 1 Col 2
Row 2 Col 0	Row 2 Col 1	Row 2 Col 2

8.3 Adding a Header

A header is a subordinate Table object. The header can be accessed through the table's *HeaderTable* attribute.

The header table is created and retrieved with the *KlgPfTableGetHeaderTable()* method. It is passed the main Table object. Once created, you can use regular Table object methods to configure and populate the header table.

Note: Headers inherit the border styles of the body table.

```
1.    KlgPfObject table, header_table;
2.    static String celltext[] = {"Row 0 Col 0", "Row 0 Col 1", "Row 0 Col 2"};
3.    static double columns[] = {1.25, 1.25, 2.25, -1.0};
4.    int ncolumns = 3;

5.    /* print some text in the flow */
6.    KlgPfFlowPrint(doc, "Simple table example:  this program demonstrates");
      KlgPfFlowPrint(doc, "construction of a simple table using PageFormatter.");

7.    /* create a table with three columns */
8.    table = KlgPfTableCreate(doc, columns, ncolumns);

9.    /* add celltext as a row */
10.   KlgPfTableAddRow(table, celltext);

11.   /* Get the header table and create the headers
12.       NOTE:  Headers do not need to be created before the table contents */
13.   header_table = KlgPfTableGetHeaderTable(table);
14.   KlgPfTablePrintToCell(header_table, 0, 0, "Column 0");
15.   KlgPfTablePrintToCell(header_table, 0, 1, "Column 1");
16.   KlgPfTablePrintToCell(header_table, 0, 2, "Column 2");

17.   /* render the table in the flow and destroy the table object */
18.   KlgPfTablePrintAndDestroy(doc, table);
```

Figure 27 Flowing a simple table

If the code was added to a PageFormatter program, you would get the following output:

Column 0	Column 1	Column 2
Row 0 Col 0	Row 0 Col 1	Row 0 Col 1

8.4 Changing the Table Appearance

The table object provides a number of attributes that can be set to control its appearance.

8.4.1 Alignment

The Table object's `Alignment` attribute defines the table's alignment within the frame. By default PageFormatter centers the entire table in the current column of the current frame. You can change the table's horizontal alignment with *KlgPfTableSetAlignment()*.

For example:

```
KlgPfTableSetAlignment(table, KLGPF_ALIGN_RIGHT);
```

KlgPfTableSetAlignment() is passed the Table object and an alignment constant. Valid alignments are:

KLGPF_ALIGN_LEFT	The table is left aligned.
KLGPF_ALIGN_CENTER	The table is centered.
KLGPF_ALIGN_RIGHT	The table is right aligned.
KLGPF_JUSTIFY	Behaves like KLGPF_ALIGN_CENTER.

8.4.2 Background Color

The Table object's BackgroundColor attribute defines the table's background color. By default this is transparent. You can apply a Color object to the table's background with *KlgPfTableSetBackgroundColor()*.

For example:

```
backgroundcolor = KlgPfColorCreateGrey(0.5)
KlgPfTableSetBackgroundColor(table, backgroundcolor);
```

KlgPfTableSetBackgroundColor() is passed the Table object and a Color object.

Unless explicitly set, Row, Column, and Cell objects inherit this attribute.

8.4.3 CellStyle

The Table object's CellStyle attribute defines the Style object applied to text rendered with the *KlgPfTableAddRow(), KlgPfTablePrintToCell()*, and *KlgPfTableAddRange()* methods.

You can set a table's CellStyle attribute with the *KlgPfTableSetCellStyle()* method. The method is passed the Table object and a Style object.

Unless explicitly set, Row, Column, and Cell objects inherit this attribute.

Modifying our code from Figure 25, we will use the *KlgPfTableRowSetCellStyle()* method in Figure 28 to render the first row of the table in bold and center the text in the cell's frame:

```
1.    KlgPfObject table, curr_style, row0, row0_style;
2.    static double columns[] = {1.00, 1.25, 2.25, -1.0};
3.    double ncolumns = 3;

4.    /* print some text in the flow */
5.    KlgPfFlowPrint(doc, "Simple table example:  this program demonstrates");
6.    KlgPfFlowPrint(doc, "construction of a simple table using PageFormatter.");

7.    /* create a table with three columns */
8.    table = KlgPfTableCreate(doc, columns, ncolumns);

9.    /* make a copy of the document's current style and modify it */

10.   curr_style = KlgPfDocGetCurrentStyle(doc);
11.   row0_style = KlgPfStyleCopy(curr_style, doc, "Row 0 Style");
12.   KlgPfStyleSetFontStyle(row0_style, KLGPF_FONT_BOLD);
13.   KlgPfStyleSetAlignment(row0_style, KLGPF_ALIGN_CENTER);

14.   /* Get Row 0; getting Row 0 creates it */
15.   row0 = KlgPfTableGetRow(table, 0);

16.   /* Set Row 0's CellStyle */
17.   KlgPfTableRowSetCellStyle(row0, row0_style);

18.   /* populate table with KlgPfTablePrintToCell() */
19.   KlgPfTablePrintToCell(table, 0, 0, "Row 0 Col 0");
20.   KlgPfTablePrintToCell(table, 0, 1, "Row 0 Col 1");
21.   KlgPfTablePrintToCell(table, 0, 2, "Row 0 Col 2");
22.   KlgPfTablePrintToCell(table, 1, 0, "Row 1 Col 0");
23.   KlgPfTablePrintToCell(table, 1, 1, "Row 1 Col 1");
24.   KlgPfTablePrintToCell(table, 1, 2, "Row 1 Col 2");
25.   KlgPfTablePrintToCell(table, 2, 0, "Row 2 Col 0");
26.   KlgPfTablePrintToCell(table, 2, 1, "Row 2 Col 1");
27.   KlgPfTablePrintToCell(table, 2, 2, "Row 2 Col 2");

28.   /* render the table in the flow and destroy the table object */
29.   KlgPfTablePrintAndDestroy(doc, table);
```

Figure 28 Modifying a row's CellStyle

If the code was added to a PageFormatter program, you would get the following output:

Row 0 Col 0	Row 0 Col 1	Row 0 Col 2
Row 1 Col 0	Row 1 Col 1	Row 1 Col 2
Row 2 Col 0	Row 2 Col 1	Row 2 Col 2

8.4.4 Cell Alignment

The Table object's `CellAlignment` attribute defines the vertical alignment of cell text for each cell in the table. You can set alignment with *KlgPfTableSetCellAlignment()*.

For example:

```
KlgPfTableSetCellAlignment(table, KLGPF_CELL_ALIGNMENT_CENTER);
```

KlgPfTableSetCellAlignment() is passed the Table object and an alignment constant. Valid alignments are:

KLGPF_CELL_ALIGNMENT_NONE	Equivalent to KLGPF_CELL_ALIGNMENT_TOP.
KLGPF_CELL_ALIGNMENT_TOP	Text is rendered at the top of the cell.
KLGPF_CELL_ALIGNMENT_CENTER	Text is rendered in the middle of the cell.
KLGPF_CELL_ALIGNMENT_BOTTOM	Text is rendered at the bottom of the cell.

Unless explicitly set, Row, Column, and Cell objects inherit this attribute.

8.4.5 Row and Cell Height

By default row height is dynamic. A row's height is based on the cell in the row with the greatest height. By default cell height is also dynamic. A cell's height will adjust to accommodate the text in the cell. You can, however, fix a cell's height with the *KlgPfTableCellSetHeight()* method. If you explicitly set a cell height, the cell will no longer dynamically compute its size.

8.4.6 Internal Margins

The Table object defaults to a 1 point margin of white space inside each cell. Internal margins are specified with the *MarginTop*, *MarginBottom*, *MarginRight*, and *MarginLeft* attributes. You can use the *KlgPfTableSetCellMarginBottom()*, *KlgPfTableSetCellMarginTop()*, *KlgPfTableSetCellMarginRight()*, and *KlgPfTableSetCellMarginLeft()* set the attributes.

You can also use the Row, Column, and Cell objects to set internal margins specific to a particular row, column, or cell.

8.4.7 Controlling Overflow

Tables, their dimensions, and their contents should be thoughout ahead of time, ensuring the table sits on the page in a balanced and readable fashion. There may be circumstances where a table's column might extend over the page's right margin. By default, PageFormatter will print as much of the table that fits across the page and

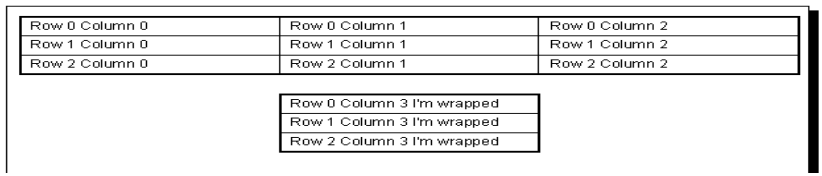
clip the rest. You may find it preferable to have the overflow columns "wrap," that is render on the next line instead of being clipped.

The Table object's *OverflowMode* attribute controls PageFormatter's behavior should a column extend over the page's right margin. You can set *OverflowMode* with *KlgPfTableSetOverflowMode()*.

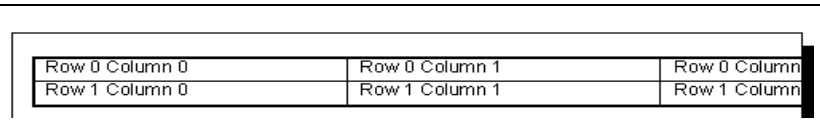
For example:

```
KlgPfTableSetOverflowMode(table, KLGPF_TABLE_OVERFLOW_WRAP_COLUMNS);
```

KlgPfTableSetOverflowMode() is passed the Table object and an overflow constant. Valid overflow mode constants are:



KLGPF_TABLE_OVERFLOW_WRAP_COLUMNS
PageFormatter renders the overflow columns on the next line after the table.



KLGPF_TABLE_OVERFLOW_CLIP_COLUMNS
PageFormatter prints as much of the table that fits across the page and clips the rest.

8.5 Borders

By default PageFormatter renders the entire table with a single black border. To alter the border appearance, you need to:

- Create a Border object.
- Set the Border object's attributes.
- Pass the border object to *KlgPfTableSetBorder()*, which sets the Table object's specified border attributes.

The sample code below demonstrates setting a border.

```
1.    KlgPfObject table, header_table, border;
2.    static String celltext[] = {"Row 0 Col 0", "Row 0 Col 1", "Row 0 Col 2"};
3.    static double columns[] = {1.25, 1.25, 1.25, -1.0};

4.    /* print some text in the flow */
5.    KlgPfFlowPrint(doc, "Simple table example:  this program demonstrates
6.    construction of a simple table using PageFormatter.");

7.    /* create a table with three columns */
8.    table = KlgPfTableCreate(doc, columns, 3);

9.    /* add celltext as a row */
10.   KlgPfTableAddRow(table, celltext);

11.   /* Get the header table and create the headers
12.   NOTE:  Headers do not need to be created before the table contents */
13.   header_table = KlgPfTableGetHeaderTable(table);
14.   KlgPfTablePrintToCell(header_table, 0, 0, "Column 0");
15.   KlgPfTablePrintToCell(header_table, 0, 1, "Column 1");
16.   KlgPfTablePrintToCell(header_table, 0, 2, "Column 2");

17.   /* Create a Border object */
18.   border = KlgPfTableBorderCreate(table);
19.   KlgPfTableBorderSetRuleType(border, KLGPF_TABLE_RULE_DOUBLE);
20.   KlgPfTableBorderSetWidth(border, 0.02);
21.   KlgPfTableBorderSetSpacing(border, 0.04);
22.   KlgPfTableBorderSetColor(border, KlgPfColorCreateFromString("blue"));

23.   /* Apply to defined Border object to the external borders of the table */
24.   KlgPfTableSetBorder(table, KLGPF_TABLE_BORDER_TOP |
                                 KLGPF_TABLE_BORDER_BOTTOM |
                                 KLGPF_TABLE_BORDER_LEFT |
                                 KLGPF_TABLE_BORDER_RIGHT, border);

25.   /* Destroy the border object */
26.   KlgPfTableBorderDestroy(border);
27.   /* render the table in the flow and destroy the table object */
28.   KlgPfTablePrintAndDestroy(doc, table);
```

Figure 29 Setting a Table's border attributes

If the above code was added to a PageFormatter program, you would get the following output:

Column 0	Column 1	Column 2
Row 0 Col 0	Row 0 Col 1	Row 0 Col 1

Creating the Border Object

A border object is created on line 18 with the *KlgPfTableBorderCreate()* method. It is passed the parent Table object.

Setting RuleType

On line 19, we define the border's ruling line by passing the *KlgPfTableBorderSetRuleType()* method the Border object and a `KlgPfTableRuleType` constant. Valid rule types are:

`KLGPF_TABLE_RULE_NONE`	No rule
`KLGPF_TABLE_RULE_SINGLE`	————————————
`KLGPF_TABLE_RULE_DOUBLE`	═══════════════════
`KLGPF_TABLE_RULE_BROKEN`	- -

Setting the Border's Width

On line 20, the width of the ruling lines is set with the *KlgPfTableBorderSetWidth()* method. The method is passed the border object and a width in document units. In this example, we're assuming document units is in inches.

Setting Spacing

In the case of a double ruling line, we also want to set the space between the rules. This is set with *KlgPfTableBorderSetSpacing()* on line 21. The method is passed the border object and spacing between the rules in document units. In this example, we're assuming document units is in inches.

If RuleType was `KLGPF_TABLE_RULE_BROKEN`, the spacing amount passed to *KlgPfTableBorderSetSpacing()* would control the length of the rule's dashes and the white space between the dashes.

Setting Border Color

A Color object can be assigned to the Border object's `Color` attribute. On line 22, we use the *KlgPfTableBorderSetColor()* method to create a Color object and set the Border object's `Color` attribute.

Setting Table Borders

Between lines 18 and 22, we created and defined a border object. This border object can now be assigned to the Table's border attributes with the *KlgPfTableSetBorder()* method. *KlgPfTableSetBorder()* is passed a Table object, the desired border locations, and a border object.

Valid `KlgPfTableBorderLocation` constants are:

Constant	Sets Attribute	Description
KLGPF_TABLE_BORDER_BOTTOM	BorderBottom	Border object applied to the table's bottom external border.
KLGPF_TABLE_BORDER_COLUMN	BorderColumn	Border object applied to the table's internal vertical borders.
KLGPF_TABLE_BORDER_HEADER	BorderHeader	Border object applied to the header table border.
KLGPF_TABLE_BORDER_LEFT	BorderLeft	Border object applied to the table's left-side external border.
KLGPF_TABLE_BORDER_RIGHT	BorderRight	Border object applied to the table's right-side external border.
KLGPF_TABLE_BORDER_ROW	BorderRow	Border object applied to the table's internal horizontal borders.
KLGPF_TABLE_BORDER_TOP	BorderTop	Border object applied to the table's top external border.

Below is an illustration of the relation between the border attributes and a rendered table:

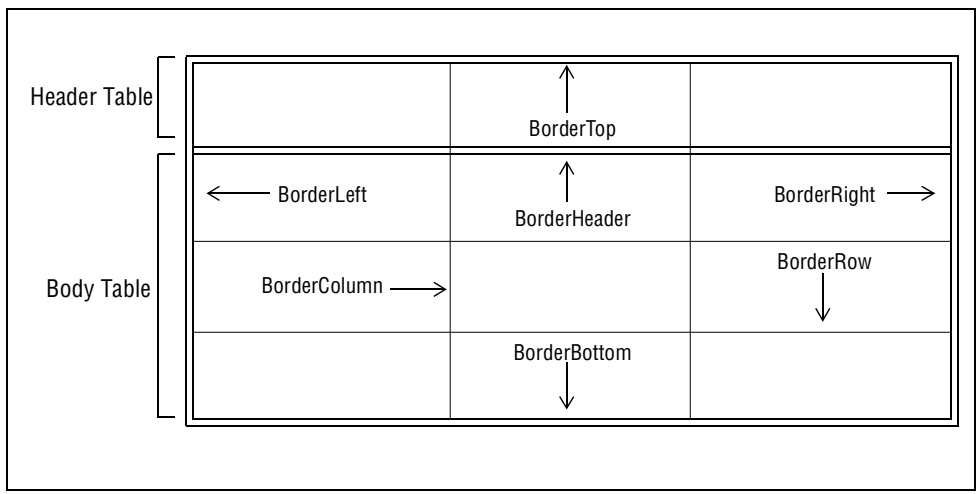


Figure 30 Border Attributes and Their Location in a Table

8.5.1 Controlling Border Appearance at Page Breaks with BorderMode

If a table does not fit vertically in the column, PageFormatter will break the table and render the remaining rows in the next flow frame. If a table has a header table, it will render the header in the new flow frame before the remaining rows.

The Table object's `BorderMode` attribute controls what border attribute is used to render the table's bottom before the break and the table's top after the break.

By default, `BorderMode` is set for `KLGPF_TABLE_MODE_EXTERNAL_BORDERS`, which indicates the bottom and top borders at the break should be rendered using the Table object's `BorderBottom` and `BorderTop` border attributes.

Using the *KlgPfTableSetBorderMode()* method, you can set BorderMode to `KLGPF_TABLE_MODE_INTERNAL_BORDERS`, which indicates the bottom and top borders at the break should be rendered using the Row object's `BorderBottom` border attribute (or if set, a cell's `BorderBottom` attribute).

Column Wrapping

The Table object's `BorderMode` attribute also controls what border attribute is used to render the table's right and left borders in the event of column wrapping. `KLGPF_TABLE_MODE_INTERNAL_BORDERS` indicates the column borders should be rendered using the `BorderColumn` attribute (or if set, a cell's `BorderRight`/`BorderLeft` attribute). `KLGPF_TABLE_MODE_EXTERNAL_BORDERS` indicates the column borders should be rendered using the `BorderRight` and `BorderLeft` attributes.

8.6 Table Inheritance

As we've seen in previous examples, you can create simple tables by only defining the Table object. As data is entered into the table via methods like *KlgPfTableAddRow()*, PageFormatter generates Row, Column, and Cell objects as needed. Each object inherits attribute settings from its parent object. Rows and Columns have a Table object as a parent object. A Cell object has a Row object as a parent. (The attribute lists at the beginning of this chapter denote which attributes inherit their settings.)

8.7 Setting Row, Column, and Cell Attributes

At times, tables require rows, columns, and cells that differ in appearance from the main table. For example, you might wish to highlight a row with a different `RuleType` or a `BackgroundColor`. Though attributes settings are inherited from a parent, the Row, Column, and Cell objects provide set and get methods to change their attributes programmatically.

To alter individual rows, columns, and cells:

- Create the table.
- Use the *KlgPfTableGetRow()*, *KlgPfTableGetColumn()*, or *KlgPfTableGetCell()* method to create and retrieve the desired Row, Column, or Cell object you wish to alter. When you use a get method on a row or a cell that does not exist, PageFormatter will create the row or cell.
- Use the object's set methods to alter the border or background color attributes.
- Populate the table.

In Figure 25 we created the following table:

Row 0 Col 0	Row 0 Col 1	Row 0 Col 2
Row 1 Col 0	Row 1 Col 1	Row 1 Col 2
Row 2 Col 0	Row 2 Col 1	Row 2 Col 2

The following example code adds background shading to row 0 and places a double rule around the cell at Row 1/Column 1:

```
1.    KlgPfObject table, cell11, cell01, cell10, color;
2.    static double columns[] = {1.00, 1.25, 2.25, -1.0};
3.    int ncolumns = 3;

4.    /* print some text in the flow */
5.    KlgPfFlowPrint(doc, "Simple table example:  this program demonstrates");
      KlgPfFlowPrint(doc, "construction of a simple table using PageFormatter.");

6.    /* create a table with three columns */
7.    table = KlgPfTableCreate(doc, columns, ncolumns);

8.    /* Retrieve Row 0's Row object and set the background color.
      PageFormatter creates the row if it does not exist */
9.    row0 = KlgPfTableGetRow(table, 0);
10.   color = KlgPfColorCreateFromString("blue");
11.   KlgPfTableRowSetBackgroundColor(row0, color);

12.   /* Create a Border object with double ruling lines */
13.   border = KlgPfTableBorderCreate(table);
14.   KlgPfTableBorderSetRuleType(border, KLGPF_TABLE_RULE_DOUBLE);

15.   /* Retrieve the cell object at Row 1 Col 1 */
16.   cell11 = KlgPfTableGetCell(table, 1, 1);

17.   /* Outline the cell at (1, 1) */
18.   KlgPfTableCellSetBorder(cell11,
          KLGPF_TABLE_BORDER_BOTTOM | KLGPF_TABLE_BORDER_RIGHT, border);

19.   /* Cells only have right & bottom border attributes, so to change the top
20.    border of a cell we have to change the bottom border of the cell above, &
21.    similarly for the left.*/

22.   cell01 = KlgPfTableGetCell(table, 0, 1);
23.   KlgPfTableCellSetBorder(cell01, KLGPF_TABLE_BORDER_BOTTOM, border);

24.   cell10 = KlgPfTableGetCell(table, 1, 0);
25.   KlgPfTableCellSetBorder(cell10, KLGPF_TABLE_BORDER_RIGHT, border);

26.   /* add cells and rows with KlgPfTablePrintToCell() */
27.   KlgPfTablePrintToCell(table, 0, 0, "Row 0 Col 0");
28.   KlgPfTablePrintToCell(table, 0, 1, "Row 0 Col 1");
29.   KlgPfTablePrintToCell(table, 0, 2, "Row 0 Col 2");
30.   KlgPfTablePrintToCell(table, 1, 0, "Row 1 Col 0");
31.   KlgPfTablePrintToCell(table, 1, 1, "Row 1 Col 1");
32.   KlgPfTablePrintToCell(table, 1, 2, "Row 1 Col 2");
33.   KlgPfTablePrintToCell(table, 2, 0, "Row 2 Col 0");
34.   KlgPfTablePrintToCell(table, 2, 1, "Row 2 Col 1");
35.   KlgPfTablePrintToCell(table, 2, 2, "Row 2 Col 2");

36.   KlgPfTableBorderDestroy(border);
37.   KlgPfColorDestroy(color);

38.   /* render the table in the flow and destroy the table object */
39.   KlgPfTablePrintAndDestroy(doc, table);
```

Figure 31 Changing Row and Cell Attributes

If the above code was added to a PageFormatter program, you would get the following output:

Row 0 Col 0	Row 0 Col 1	Row 0 Col 2
Row 1 Col 0	Row 1 Col 1	Row 1 Col 2
Row 2 Col 0	Row 2 Col 1	Row 2 Col 2

Note: The Table object's outside borders cannot be overridden at the row, cell, or column level. Setting a double border for cell (2,1)'s bottom border would not produce a double border along cell (2,1)'s bottom edge.

8.7.1 Direction Priority

PageFormatter always gives Cell attributes priority over similar Row, Column, or Table attributes. For example, if a specific cell object's `MarginBottom` has been set, the cell will be rendered using its `MarginBottom` attribute, not the Row, Column, or Table object's `CellMarginBottom` attribute.

Conflicts arise when one sets a similar attribute for a Row and Column object and they intersect (assuming no cell attribute has been set for the actual Cell object at the Row/Column intersection).

For example, in Figure 31, we set row 0 to render cells in the row with a blue background (that is, the row's `BackgroundColor` attribute is "blue"). We could also set Column 1 to render cells in the column with a red background (that is, the Column's `BackgroundColor` attribute is set as "red"). A conflict arises when PageFormatter renders the cell at (0, 1). Since cell (0, 1) is in row 0, it is to be rendered blue. But it's also in Column 1 and it's to be rendered red.

DirectionPriority Attribute

PageFormatter resolves these conflicts using the Table object's `DirectionPriority` attribute. The `DirectionPriority` attribute defaults to `KLGPF_TABLE_PRIORITY_ROW`, which gives priority to the Row object and renders the cell based on the Row object's attribute setting. Using the *KlgPfTableSetPriorityDirection()* method, you can change the `DirectionPriority` attribute to `KLGPF_TABLE_PRIORITY_COLUMN`, which gives priority to the Column object and renders the cell based on the Column object's attribute setting.

8.8 Creating a Cell Span

PageFormatter lets you create Cell objects that span several rows and/or columns. In the example below, the table's top left cell spans the entire row.

This is Row 0 Col 0, which spans cells at (0,1) and (0,2)		
Row 1 Col 0	Row 1 Col 1	Row 1 Col 2

To create a table with a span, we could use the following code:

```
1.    KlgPfObject table;
2.    static String celltext2[] = {"Row 1 Col 0", "Row 1 Col 1", "Row 1 Col 2"};
3.    static double columns[] = {1.00, 1.25, 2.50, -1.0};
4.    int start_row = 0;
5.    int start_column = 0;
6.    span_rows = 1;
7.    span_columns = 3;

8.    /* print some text in the flow */
9.    KlgPfFlowPrint(doc, "Simple table example:  this program demonstrates
      construction of a simple table using PageFormatter.");

10.   /* create a table with three columns */
11.   table = KlgPfTableCreate(doc, columns, 3);

12.   /* set cell (0,0) to span (0,0), (0,1), (0,2)\
13    Set the span before populating the table */
14.   KlgPfTableSpanCells(table, start_row, start_column,
          span_rows, span_columns);

15.   /* populate row 0 span with KlgPfTablePrintToCell() */
16.   KlgPfTablePrintToCell(table, 0, 0, "This is Row 0 Col 0, which spans
          cells at (0,1) and (0,2);
17.   /* add row 1 */
18.   KlgPfTableAddRow(table, celltext2);

19.   /* render the table in the flow and destroy the table object */
20.   KlgPfTablePrintAndDestroy(doc, table);
```

Figure 32 Creating a Cell Span

On line 14, we create the span before we populate the table. Since row 0 only has one spanned cell, we can populate that cell with the *KlgPfTablePrintToCell()* method. Row 1 can be populated with the *KlgPfTableAddRow()* method.

Note: spans cannot intersect.

9

Printer Object

Printer Object Attributes ■ *Creating a Printer Object* ■ *PostScript Printer*
PCL Printer ■ *Printing Adobe Acrobat PDF Files* ■ *Printing to a Stream*
Printing to a File ■ *Output Policy* ■ *Callbacks*
Print Previewer ■ *Print Dialog Object*

The Printer object provides attributes and methods that control how the rendered document is printed on an output device: the number of pages to output, the number of copies, whether copies should be collated, and so on. The Printer object, when created, also generates a list of Font Family and Font objects based on provided printer drivers. Provided printer drivers also supply the Printer object with information about printer options such as what paper sizes are supported (for example “Letter”, “Legal”, “A4”, and so on) and what resolutions are supported.

9.1 Printer Object Attributes

Attribute	Data Type
Collate	Boolean
CurrentDocument	KlgPfObject
Display	KlgPfObject
Filename	String
FontFamilyList	KlgPfObject
FontList	KlgPfObject
Name	String
NumCopies	int
OutputCallback	KlgPfCallbackRecord
OutputPageEnd	int

Attribute	Data Type
OutputPageStart	int
OutputType	KlgPfIOType
PaperSizeUnits	KlgPfUnit
PrinterOptionsList	KlgPfObject
PrinterOptionSelectedList	KlgPfObject
PrinterType	KlgPfPrinterTypeEnum
ResolutionSupportedList	KlgPfObject
ReverseOrder	Boolean
Stream	FILE
Type	XtPointer

9.2 Creating a Printer Object

There are four types of Printer objects:

- A PostScript printer (include <KlgPfPS.h>)
- Adobe Acrobat PDF printer (include <KlgPfAcro.h>)
- A PCL printer (include <KlgPfPCL.h>)
- An X11 Window printer (used by the PageFormatter Print Preview widget) (include <KlgPfX11.h>)

9.2.1 Setting Printer Attributes

For a complete list of Printer object attributes and set and get methods, see Appendix K on page 221. Below we will review some of the more common attributes you might want to set at run time.

9.2.2 Setting Multiple Copies

If you wish PageFormatter to output more than one copy of the rendered document, you can set the NumCopies attribute with the *KlgPfPrinterSetNumCopies()* method.

9.2.3 Collating

By default the PageFormatter will not collate multiple copies. In other words, it will print multiple copies of each page before proceeding to the next. You can have PageFormatter collate copies by setting the Collate attribute to True with the *KlgPfPrinterSetCollate()* method.

9.2.4 Reverse Order Printing

You can specify reverse order printing (printing the last page first) by setting the Printer object's `ReverseOrder` attribute to `True` with the *KlgPfPrinterSetReverseOrder()* method.

9.2.5 Printing a Range of Pages

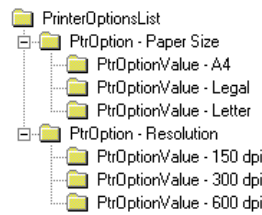
You do not have to output the entire rendered document. You can specify a range of pages by specifying a start and an end page. The start page is specified by setting the `OutputPageStart` attribute with the *KlgPfPrinterSetOutputPageStart()* method.

An end page is specified by setting the `OutputPageEnd` attribute with the *KlgPfPrinterSetOutputPageEnd()* method.

If you specify a start page without specifying an end page, PageFormatter will print from the start page to the end of the document. If you specify an end page without specifying a start page, PageFormatter will print from the first page in the document to the end page.

9.2.6 Printer Options

When you create a PostScript or a printer, PageFormatter generates a list of printer options (In the case of PostScript, these are defined in conjunction with the PPD file). Each option is defined in PageFormatter by a PtrOption object. These PtrOption objects are added to the Printer object's `PrinterOptionsList` list.



Tree diagram of how PtrOption and PtrOptionValue objects are organized under PrinterOptionsList

Each PtrOption object is comprised of a list of PtrOptionValue objects. Each PtrOptionValue object represents a possible user selection. PtrOptionValue objects are stored in the PtrOption object's `ValueList` list.

For example, the PtrOption object representing the PageSize option may have a "Letter" PtrOptionValue object, a "Legal" PtrOptionValue object, an "A4" PtrOptionValue object, and so on.

Selected PtrOptionValue objects are stored in the Printer object's `PrinterOptionSelectedList`. For example, `PrinterOptionSelectedList` might contain a "Letter" PtrOptionValue object and a "600 dpi" PtrOptionValue object. An PtrOptionsValue object may be set with the *KlgPfPrinterSetPrinterOption()* method.

For more information on the PtrOption object and the PtrOptionValue object see Appendix M on page 237.

9.3 PostScript Printer

To create a PostScript printer include the <KlgPfPS.h> header file and call the *KlgPfPrinterCreatePS()* method.

Note: Including the header file <KlgPfAll.h> does not include: <KlgPfPS.h>, <KlgPfAcro.h>, <KlgPfPCL.h> or <KlgPfX11.h>. You will have to include these headers individually.

9.3.1 PPD File

PageFormatter supplies a number of PPD (PostScript Printer Description) files that describe what PostScript features are supported by each printer. The directory where these files are located is set using the *KlgPfGlobalSet()* function. By default, the files are located in *$KLGPRHOME/lib/ps/ppd*. You can set where PageFormatter looks for PPD files by using *KlgPfGlobalSet()* with the type KLGPF_GLOBAL_PPD_DIR and the new directory name. A PPD file provides a list of options for a particular printer and the code to enable those options.

PageFormatter is made aware of what PPD file to parse by the *printers* file in *$KLGPRHOME/lib/ps*. A different file can be specified by using *KlgPfGlobalSet()* with the type KLGPF_GLOBAL_PRINTERS_FILE and the desired filename. In the *printers* file you can insert the name of the printer and its corresponding PPD file. For example:

```
ps hp5m1_v1.ppd
ps2 hp4m3_v1.ppd
```

When you create a PostScript printer object, PageFormatter will scan the *printers* file for a printer name matching your PRINTER environment variable. PageFormatter will then search for the indicated PPD file in a user-specified directory, the directory specified in the PPD_DIR environment variable, or one of the following directories: *$KLGPRHOME/lib/ps/ppd*, */usr/lib, /usr/lib/ps*, */usr/local/lib*, and */usr/local/lib/ps*. If it finds the indicated PPD file, it creates the appropriate PtrOption and PtrOptionValue objects.

If PageFormatter does not provide a PPD file for your printer, you can usually obtain one from your printer manufacturer/distributor, your printer manufacturer/distributor's web site, or from Adobe's web site (*http://www.adobe.com*).

9.4 PCL Printer

To create a PCL printer include the <KlgPfPCL.h> header file and call the *KlgPfPrinterCreatePCL()* method.

Note: Including the header file <KlgPfAll.h> does not include: <KlgPfPCL.h>, <KlgPfPS.h>, <KlgPfAcro.h>, or <KlgPfX11.h>. You will have to include these headers individually.

9.5 Printing Adobe Acrobat PDF Files

Adobe's Acrobat Portable Document Format (PDF) allows you to generate compact files readable across various platforms that retain their formatting characteristics. PageFormatter provides a printer type that allows you to output your Document object to a PDF file.

To create a Printer object that will print to a PDF file, include the `<KlgPfAcro.h>` header file and call the *KlgPfPrinterCreateAcro()* method.

Note: Including the header file `<KlgPfAll.h>` does not include: `<KlgPfPS.h>`, `<KlgPfAcro.h>`, `<KlgPfPCL.h>` or `<KlgPfX11.h>`. You will have to include these headers individually.

To print a PDF, you need to set the Printer object's `OutputType` attribute with the *KlgPfPrinterSetOutputType()* method and the `filename` attribute with the *KlgPfPrinterSetFilename()* method.

For example:

```
acroprinter = KlgPfPrinterCreateAcro("Acrobat Printer");
doc = KlgPfDocCreate(acroprinter, "doc");
KlgPfPrinterSetOutputType(acroprinter, KlgPfIOFilename);
KlgPfPrinterSetFilename(acroprinter, "output.pdf");
KlgPfPrinterOutput(acroprinter, doc);
```

The *KlgPfPrinterCreateAcro()* method creates a printer. The *KlgPfPrinterSetOutputType()* method is passed the Printer object and a `KlgPfIOType` input/output type. Valid values are:

`KlgPfIOFilename`	Print to an unopened file specified by the `Filename` attribute.
`KlgPfIOStream`	Print to a writable stream. This is the default.

The *KlgPfPrinterSetFilename()* method is called to set the Printer object's `Filename` attribute. It is passed the Printer object and the name of a file.

9.6 Printing to a Stream

By default, PageFormatter will send output from a PostScript, HP PCL or Acrobat PDF printer object to a stream. The stream PageFormatter sends output to is defined by the Printer object's `Stream` attribute. It is `NULL` by default. To set the `Stream` attribute, call *KlgPfPrinterSetStream()*.

For example:

```
FILE *fp;
fp = fopen(OUTPUT_FILE, "w");
psprinter = KlgPfPrinterCreatePS("postscript_printer");
doc = KlgPfDocCreate(psprinter, "doc");
KlgPfPrinterSetOutputType(psprinter, KlgPfIOStream);
KlgPfPrinterSetStream(psprinter, fp);
KlgPfPrinterOutput(psprinter, doc);
```

9.7 Printing to a File

Output can be channeled to a named file. To print to a file you need to set the Printer object's `OutputType` attribute with the *KlgPfPrinterSetOutputType()* method and the `filename` attribute with the *KlgPfPrinterSetFilename()* method.

For example:

```
psprinter = KlgPfPrinterCreatePS("postscript_printer");
doc = KlgPfDocCreate(psprinter, "doc");
KlgPfPrinterSetOutputType(psprinter, KlgPfIOFilename);
KlgPfPrinterSetFilename(psprinter, "output.ps");
KlgPfPrinterOutput(psprinter, doc);
```

The *KlgPfPrinterCreatePS()* method creates a printer. The *KlgPfPrinterSetOutputType()* method is passed the Printer object and a `KlgPfIOType` input/output type.

The *KlgPfPrinterSetFilename()* method is called to set the Printer object's `Filename` attribute. It is passed the Printer object and the name of a file.

9.8 Output Policy

By default PageFormatter completely renders the document before passing it to the Printer object. When speed is an issue, you can change the Document object's `OutputPolicy` attribute from `KLGPF_OUTPUT_AS_REQUESTED` to `KLGPF_OUTPUT_ASAP` with the *KlgPfDocSetOutputPolicy()* method. When `OutputPolicy` is set to `KLGPF_OUTPUT_ASAP`, pages will be passed to the Printer when the page is fully rendered (that is, when the page's `LayoutStatus` is `KLGPF_LAYOUT_COMPLETE`) and the flow transferred to the next page.

Note: PageFormatter will change the `OutputPolicy` to `KLGPF_OUTPUT_AS_REQUESTED` if it encounters a page with macros that cannot be resolved until all pages are completely rendered.

9.9 Callbacks

The Printer object provides an `OutputCallback` attribute. The `OutputCallback` attribute specifies a procedure (along with a pointer to any user data) that should be called when an output process generates a callback. When PageFormatter generates an output callback, it passes the following structure to the callback procedure:

```
typedef struct {
   KlgPfCallbackReason reason;
   KlgPfObject         document;
   KlgPfObject         printer;
} KlgPfOutputCallbackStruct
```

The *reason* member is one of the following enumerated types:

`KLGPF_REASON_OUTPUT_NEW`	PageFormatter begins an output process.

KLGPF_REASON_OUTPUT_PAGE	PageFormatter outputs a page.
KLGPF_REASON_OUTPUT_END	PageFormatter ends an output process.

9.9.1 Defining Procedures

The procedure must be defined to be the following type:

```
typedef void (*KlgPfCallbackProc)(
   XtPointer         cbs,
   XtPointer         call_data
   );
```

9.9.2 Registering Procedures

After you've created such a procedure, you need to make PageFormatter aware of its existence (and any user data that should be passed to the callback procedure) by setting the Printer object's `OutputCallback` attribute. The `OutputCallback` attribute is set with *KlgPfPrinterSetOutputCallback()*.

9.10 Print Previewer

If you are developing under X11, PageFormatter provides methods for the creation of an on-screen Preview widget that allows you to view your output in an X11 window.



Figure 33 Print Preview Widget (within a dialog shell)

9.10.1 Print Preview Object Attributes

Attribute	Type
CurrentPage	int
Document	KlgPfObject
Height	int
Printer	KlgPfObject
Width	int

9.10.2 Creating a Print Preview Widget

The Print Preview widget is created with the *KlgPfXmPreviewCreate()* method. Include the `<KlgPfX11.h>` header file.

Note: Including the header file `<KlgPfAll.h>` does not include: `<KlgPfPS.h>`, `<KlgPfAcro.h>`, `<KlgPfPCL.h>` or `<KlgPfX11.h>`. You will have to include these headers individually.

For example:

```
preview = KlgPfXmPreviewCreate(toplevel, 154, 200);
```

The method is passed a parent widget, a width value in pixels, and a height value in pixels. It returns a Print Preview widget. If either the width or height is passed a value less than or equal to zero, a default width or height is calculated.

This method is a convenience function that creates all component widgets of the Print Preview widget as well as creating an X window printer with the *KlgPfPrinterCreateX11()* method.

9.10.3 Displaying the Print Preview Widget

The Print Preview widget can be displayed with the *KlgPfXmPreviewPopup()* method.

9.10.4 Hiding the Print Preview Widget

The Print Preview widget can be hidden or destroyed with the *KlgPfXmPreviewPopdown()* method. The method is passed the Print Preview widget as well as a Boolean value. If `True`, the object is destroyed. If `False`, the object is merely hidden.

9.10.5 Updating the Print Preview Widget

To display the current page of the document in the Print Preview widget, the *KlgPfXmPreviewUpdate()* method is called. This method is called whenever *KlgPfXmPreviewPopup()* is called.

9.10.6 Setting A Document

The Print Preview widget will preview the Document object specified by the `Document` attribute. To set the `Document` attribute, call the *KlgPfXmPreviewSetDocument()* method. This method calls *KlgPfXmPreviewUpdate()* automatically.

9.10.7 Setting the Current Page

The page displayed is based on the `CurrentPage` attribute. The Preview widget provides Previous and Next buttons that change the `CurrentPage` attribute. You may wish to modify the widget and provide the end users with alternative means of altering the `CurrentPage` attribute. To do so, use the *KlgPfXmPreviewSetPageNumber()* method.

9.11 Print Dialog Object

The Print Dialog object provides methods for creating a Motif Widget that lets you provide the end user with a GUI printer dialog box, which controls PostScript output to a line printer or a file. The Print Dialog's on-screen controls correspond to the Printer object's settable attributes. When controls are changed using the dialog box, appropriate changes are made to the Printer object attributes.

9.11.1 Creating a Print Dialog Widget

The Print Dialog object is created with the *KlgPfXmPrintDialogCreate()* method. This method is a convenience function that creates all component widgets of the print dialog. The Printer object it requires as an argument should be a PostScript printer (as a line printer expects PostScript input).

9.11.2 Displaying the Print Dialog Widget

The print dialog can be displayed with the *KlgPfXmPrintDialogPopup()* method.

9.11.3 Hiding the Print Dialog Widget

The print dialog can be hidden or destroyed with the *KlgPfXmPrintDialogPopdown()* method. The method is passed the Print Dialog object as well as a Boolean value. If `True`, the object is destroyed. If `False`, the object is merely hidden.

9.11.4 Updating the Print Dialog Widget

The print dialog's on-screen controls correspond to the Printer object's settable attributes. If the Printer object's attributes change, you can call *KlgPfXmPrintDialogUpdate()* to update the print dialog to reflect the Printer object attributes.

9.11.5 Source Code

The source code for the Print Previewer and the Print Dialog object is in *$KLGPRHOME/src/pf/util*. The code for the Print Dialog object provides an example of how to use the PtrOption object methods.

10

Lists

Creating and Destroying a List ■ *Adding and Deleting List Items*
Searching and Retrieving List Items ■ *Sorting a List* ■ *User-Defined Data Callbacks*

The List object provides a convenient way of storing groups of items of the same size. Items can be added, deleted, sorted, and retrieved using methods defined for these purposes. The following sections describe these methods.

Note that the list specified by a List object is zero-based: the first element has index 0, the second element has index 1, and so on.

10.1 Creating and Destroying a List

To create a list, call the *KlgPfListCreate()* method:

```
list = KlgPfListCreate(sizeof(int));
```

This method is passed one parameter, which is the size of each of the elements in the list. (Elements of different types can be stored in the same list, provided they are the same size.) The created list is returned.

To destroy a list created by *KlgPfListCreate()*, call *KlgPfListDestroy()*:

```
KlgPfListDestroy(list);
```

All memory used by the List object (but not necessarily its elements) is freed.

Duplicating a List

To make a copy of an existing list, use the *KlgPfListDuplicate()* method:

```
newlist = KlgPfListDuplicate(oldlist);
```

This makes a copy of the list and all its elements, and returns a pointer to the copy. The original list is unchanged.

10.2 Adding and Deleting List Items

Adding List Items

You can add an item to an existing list in any of four places:

- At the end of a list
- Immediately preceding an existing item
- Immediately following an existing item
- In its proper (sorted order) position

To add an item to the end of an existing list, call *KlgPfListAppend()*:

```
index = KlgPfListAppend(list, newitem);
```

This method requires two parameters: the list to be added to, and the item to be added. This item must be the same size as the size specified when the list was created by *KlgPfListCreate()*. The method returns the index of the new item (with 0 representing the first element of the list, 1 the second, and so on).

To place a new item in front of an existing list item, call *KlgPfListInsertBefore()*:

```
newindex = KlgPfListInsertBefore(list, oldindex, newitem);
```

KlgPfListInsertBefore() requires three parameters: the list to be added to, the index of the existing list item (with 0 representing the first item), and the item to be added. The index of the new item is returned.

Similarly, *KlgPfListInsertAfter()* places a new item immediately after an existing list item.

Its parameters and return value are identical to those of *KlgPfListInsertBefore()*.

If your list is sorted, you can add an item in its proper sorted position by calling *KlgPfListAdd()*:

```
index = KlgPfListAdd(list, newitem);
```

This method requires two parameters: the list and the item to be added. The index of the new item is returned (this assumes you've previously set a *KlgPfListSetCompareProc*(), see Section 10.4 on page 124).

Removing and Deleting List Items

The object defines routines that remove an item from a list, and additional routines that both remove an item from a list and destroy the item.

To remove an item from a list, call *KlgPfListRemove()*:

```
retpos = KlgPfListRemove(list, position);
```

This method is passed the list and the position of the item to be removed. This position is also returned.

To remove all items from a list, call *KlgPfListRemoveAll()*.

Note that neither of the above methods actually frees the item memory. To destroy an item as well as remove it, call *KlgPfListDelete()*:

```
retpos = KlgPfListDelete(list, position);
```

This method, like *KlgPfListRemove()*, is passed the list and the position of the item to be removed, and returns the position.

If you have defined additional data structures associated with a particular item, which need to be destroyed when the item is deleted, you can define a destroy procedure by calling *KlgPfListSetDestroyProc()*:

```
KlgPfListSetDestroyProc(list, my_destroy_proc);
```

The destroy procedure is called whenever *KlgPfListDestroy()* is called, and must be defined to be of the following type:

```
void my_destroy_proc (
    KlgPfObject         object,
    XtPointer           item,
    XtPointer           user_data
    );
```

object is the list; *item* is the item to be destroyed. *user_data* is set with *KlgPfListSetUserData()*.

The List object also enables you to destroy a KlgPfObject object in a list by calling *KlgPfListDestroyObject()*:

```
KlgPfObject             list, *objptr;
XtPointer               user_data
KlgPfListDestroyObject(list, objptr);
```

The two parameters are the list itself and a pointer to the object to be destroyed. *user_data* is set with *KlgPfListSetUserData()*.

To destroy a non-object item in a list, call *KlgPfListDestroyXtPointer()*:

```
KlgPfObject             list;
XtPointer               *item;

KlgPfListDestroyXtPointer(list, item);
```

This method is passed the list and a pointer to the item to be destroyed.

10.3 Searching and Retrieving List Items

The List object defines several methods that provide the following search and retrieval operations:

- Determining whether a KlgPfObject is a list
- Finding an item in a list
- Retrieving an indexed element of a list or the entire array of elements in the list
- Searching forward or backward in a list

Other methods enable you to obtain the number of items in the list and their size.

Checking a List

The *KlgPfListIsList()* method determines whether a KlgPfObject is actually a List object:

```
Boolean                 is_list;
KlgPfObject             object;

is_list = KlgPfListIsList(object);
```

KlgPfListIsList() returns `True` if the object is a List object, and `False` if it is not.

Finding an Item in a List

To find an item in a list, use either of the following methods:

- *KlgPfListFind()*, which finds a list item matching a specified item using the List comparison procedure
- *KlgPfListLookupItem()*, which finds a list item matching a specified item by comparing the corresponding memory

Both methods are passed the list and the item to be compared. Both return the index of the matched item if found, or `KLGPF_LIST_ITEM_NOT_FOUND` if the item is not contained in the list.

To specify the comparison procedure to be used by *KlgPfListFind()*, call *KlgPfListSetCompareProc()*:

```
KlgPfListSetCompareProc(list, my_compare_proc);
```

This comparison procedure must be defined to have the following prototype:

```
int my_compare_proc (
    XtPointer           item1,
    XtPointer           item2
    );
```

item1 and *item2* are the two items to be compared. The comparison procedure must return the following:

- a positive integer if *item1* is greater than *item2*
- 0 if the two items are equal
- a negative integer if *item1* is less than *item2*

To retrieve the comparison procedure currently in use, call *KlgPfListGetCompareProc()*:

```
KlgPfListItemCompareProc myproc;

myproc = KlgPfListGetCompareProc(list);
```

The returned procedure is of type `KlgPfListItemCompareProc`, which is defined to be the following:

```
typedef int (*KlgPfListItemCompareProc) (
    XtPointer item1, XtPointer item2
);
```

This comparison procedure is also used in sorting.

Retrieving an Item By Index

To retrieve an item given its position, call *KlgPfListGetItem()*:

```
XtPointer              item;
KlgPfObject            list;
int                    index;

item = KlgPfListGetItem(list, index);
```

KlgPfListGetItem() is passed two parameters: the list, and the position of the item to be returned (with 0 representing the first item, 1 the second, and so on). The item at this position is returned. If the position is invalid, `NULL` is returned.

To set the item at a particular position, call *KlgPfListSetItem()*:

```
KlgPfListSetItem(list, index, item);
```

This method is passed three arguments: the list, the index of the new item (with 0 representing the first item), and the new item itself. This method returns `True` if the set operation was successful, or `False` if the index specified is invalid. The following example shows how to retrieve an item using its index:

```
#include <stdio.h>
#include <stdlib.h>
#include <KlgPfList.h>
typedef struct __studentRec{
 int age;
 char address[150];
 char name[50];
}studentRec;

int main( int argc, char *argv[]){

 studentRec rec = {15, "39 Nowhere Ville", "B.O. problem"};
 studentRec getRec;  /* Record to retrieve from list */
 KlgPfObject list;   /* List of Student Records */

 list = KlgPfListCreate(sizeof(studentRec));  /* Initialize list */

 /* Observe that you pass in the address of the element
  * to append to the list, and it must be cast to an
  * XtPointer to avoid compilation warnings
  */
 KlgPfListAppend(list, (XtPointer)&rec);

 /* GetItem returns an XtPointer to the node at
  * position 0 in the list.
  */
 getRec = *(studentRec *)KlgPfListGetItem(list, 0);

 printf("Name: %s - Address: %s - Age: %d\n", getRec.name,
             getRec.address,
             getRec.age);
/* Lists which the user creates (as opposed to
  * those lists which the user gets using KlgPfGetList...)
  * should be destroyed to reclaim memory
  */
 KlgPfListDestroy(list);
return 0;
}
```

Retrieving the List Array

To retrieve the entire array of list elements, call *KlgPfListGetItems()*:

```
KlgPfObject             list;
XtPointer               array;

array = KlgPfListGetItems(list);
```

If the list is empty, `NULL` is returned.

Searching for an Item

To search forward in a list for a specified item, call *KlgPfListSearchForward()*:

```
int                     pos;
KlgPfObject             list;
int                     start_position;
XtPointer               item;

pos = KlgPfListSearchForward(list, start_position, item);
```

This method requires three parameters: the list, the position in the list at which to start searching (0 represents the first item), and the item to search for. If the item is matched, the position of the item in the list is returned; otherwise, `KLGPF_LIST_ITEM_NOT_FOUND` is returned.

To search backward in a list for a specified item, call *KlgPfListSearchBackward()*:

```
pos = KlgPfListSearchBackward(list, start_position, item);
```

The parameters and return value are the same as those of *KlgPfListSearchForward()*.

Retrieving List Information

To obtain the number of items in a list, call *KlgPfListGetItemCount()*:

```
int                     numitems;
KlgPfObject             list;

numitems = KlgPfListGetItemCount(list);
```

This method is passed the list as its argument.

To obtain the size of an item in the list, call *KlgPfListGetItemSize()*:

```
int                     itemsize;
KlgPfObject             list;

itemsize = KlgPfListGetItemSize(list);
```

10.4 Sorting a List

To sort the items in a list, call *KlgPfListSort()*:

```
KlgPfListSort(list);
```

The comparison procedure defined by *KlgPfListSetCompareProc()* is used to determine the sort order. If no comparison procedure is specified, the following default sort behavior is used:

- If the list items are of a size less than or equal to 8, the items are sorted in byte order
- If the size is greater than 8, no sorting is performed if no comparison procedure is defined.

To determine whether a list is sorted, call *KlgPfListIsSorted()*:

```
int                 is_sorted;

is_sorted = KlgPfListIsSorted(list);
```

This method returns a non-zero value if the list is sorted, and 0 if it is not.

Relocating List Elements

To move list elements from one location in the list to another, call *KlgPfListMoveItemsBefore()*:

```
Boolean             ok;
KlgPfObject         list;
int                 destination;
int                 source;
int                 num_items;

source = 3;         /* fourth element is the first to be moved */
num_items = 3;      /* move three elements */
destination = 0;    /* place before first element */
ok = KlgPfListMoveItemsBefore(list, destination, source,
    num_items);
```

KlgPfListMoveItemsBefore() requires four parameters:

- The list containing the items to be moved
- The index of the destination element (with 0 representing the first element); the block of elements being moved is placed immediately before this element
- The index of the first element of the block being moved
- The number of elements to be moved

KlgPfListMoveItemsBefore() returns `True` if the move was successful, or `False` if an error occurred.

10.5 User-Defined Data

The List object enables you to define user data to be associated with the list. To define user data, pass a pointer to the user data to the *KlgPfListSetUserData()* method:

```
KlgPfObject         list;
XtPointer           user_data;

KlgPfListSetUserData(list, user_data);
```

You are responsible for freeing the memory used by the user data.

To retrieve the current user data associated with a list, call *KlgPfListGetUserData()*:

```
user_data = KlgPfListGetUserData(list);
```

If no user data is defined for this list, *KlgPfListGetUserData()* returns `NULL`.

10.6 Callbacks

PageFormatter provides a callback for the list object. A list callback is generated when a list item is added, moved, removed, or destroyed.

10.6.1 Callback Structure

When callback events are generated, PageFormatter passes a callback structure containing a `KlgPfCallbackReason` indentifying what triggered the event. It passes the following structure:

```
typedef struct {
   KlgPfCallbackReason          reason;
   XtPointer                    user_data;
   XtPointer                    modified_item;
   int                          position; /* position of change */
} KlgPfListChangedCallbackStruct;
```

Reason

The *reason* member could be `KLGPF_REASON_LIST_ADD` (when an item is added), `KLGPF_REASON_LIST_DELETED` (an item is deleted), `KLGPF_REASON_LIST_MOVE` (when an item is moved), `KLGPF_REASON_LIST_PENDING_DELETE` (the item is about to be removed or destroyed, allows you a chance to double check whether you want the item deleted or not), or `KLGPF_REASON_LIST_SET` (when an old element is destroyed and a new element is copied in its place).

10.6.2 Defining Procedures

You can create a procedure that makes use of `KlgPfListChangedCallbackStruct` by creating a procedure that conforms to PageFormatter's list callback type definition:

```
typedef void (*KlgPfListChangedProc) (
   KlgPfObject list,
   KlgPfListChangedCallbackStruct *call_data
);
```

10.6.3 Registering Procedures

After you've created such a procedure, you need to make PageFormatter aware of its existence. You can register your procedure by adding it to the List object's `ChangedProc` attribute with the *KlgPfListSetChangedProc()* method.

11

Using Template Builder

Central to using PageFormatter is the document template, which determines how text flows in the document. Template Builder is a GUI-based utility that allows programmers to quickly and easily create document templates. By simply adding pages and frames to a document and setting the text flow, you can create your PageFormatter templates in just minutes.

The following sections explain how to run Template Builder and create your own templates. For more information on creating templates, see Page Object on page 37.

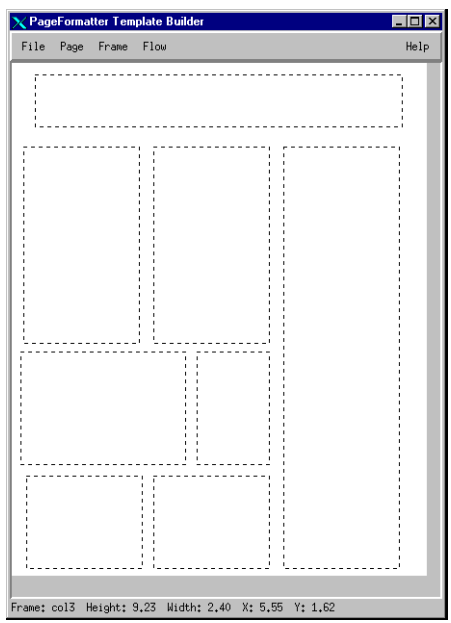


Figure 34 A document template loaded in Template Builder

11.1 Getting Started

Template Builder comes with several sample templates to help you get started.

Running Template Builder

Template Builder has an application defaults resource file that you should load before running the program. Put *$KLGPRHOME/src/template_builder/XTemplateBuilder.res* into your application defaults directory or set `XENVIRONMENT` to it.

To run template builder, `cd` to the directory where Template Builder is located (*$KLPRHOME/bin*), and at the command line type:

```
TemplateBuilder
```

Using Sample Template Files

To use the sample template files from within Template Builder:

1. Run Template Builder.
2. On the **File** menu, click **Open**. The Open dialog appears.
3. Browse to the *$KLPRHOME/src/templates* directory, or click the **Goto Templates** button.
4. In the **Directories** list, select the template file you want to open and click **OK**.

11.2 Using Template Builder

When Template Builder is launched, a new document and a first page are created. By defining and adding pages – which are made up of zero or more frames – you determine how text and images flow in your document.

Use the following procedure to create your own document templates.

11.2.1 Step 1: Setting Page Options

With an open document and first page (*page1*), the first step is to define how your page will appear. From the Page menu you can set the page color, size and the order in which they will appear in the document.

To add a new page to the document:

1. On the **Page** menu, click **New**. The New Page dialog appears.
2. Type the page name and click **OK**.

To set the page color:

1. On the **Page** menu, click **GoTo** and from the dialog select the page you want to change.

2. On the **Page** menu, click **Color**. The Color dialog appears.
3. Do one of the following:
   - Use the three RGB scrollbars to set the pages desired background color.
   - In the **Color** field, type the color name.

   If a color is specified by name, the name should appear in the X Server's color database. On Unix systems, this is normally the file */usr/lib/X11/rgb.txt*.
4. Click **OK** to apply the changes and return to Template Builder.

To set the page size:

1. On the **Page** menu, click **GoTo** and from the dialog select the page you want to change.
2. On the **Page** menu, click **Size**. The Page Size dialog appears.
3. In the **Height** and **Width** fields, type the desired page size.
4. In the **X** and **Y** fields, enter the starting page coordinates. When used in the page definition area, the page's beginning x and y coordinates are relative to the upper-left edge of the page.
5. Click **OK** to apply the changes and return to Template Builder.

To set the page order:

1. On the **Page** menu, click **Order**. The Page Order dialog appears.
2. A list box shows all the pages in the current template. To change where a page appears in the template, select it and use the **Up** and **Down** buttons.

11.2.2 Step 2: Adding and Setting Frame Options

Frames determine where text will appear on a page and the order in which it flows. Once you have added frames to a page, you can set the frame size, color, margins and columns.

To add a frame:

1. On the **Frame** menu, click **New**. The New Frame dialog appears.
2. Type the frame name and click **OK**. Notice how the mouse pointer changes on the drawing area.
3. Position the cursor where you want the top left corner of the frame, and click and drag the frame to the desired size. You can change the frame size by positioning the cursor over a frame edge and dragging the edge to the new size.

To set the frame color:

1. On the **Frame** menu, click **Color**. The Frame Color dialog appears.
2. In the Frame Colors list box, select the frame(s) you want to change.

3. Do one of the following:
   - Use the three RGB scrollbars to set the frames desired color.
   - In the **Color** field, type the color name.

   If a color is specified by name, the name should appear in the X Server's color database. On Unix systems, this is normally the file */usr/lib/X11/rgb.txt*.
4. Click **OK** to apply the changes and return to Template Builder.

To set the frame size:

1. On the Frame menu, click Size. The Frame Size dialog appears.
2. From the list, select the frame you want to change.
3. In the **Height** and **Width** fields, type the desired frame size.
4. In the **X** and **Y** fields, enter the frame's starting coordinates. When used in the page definition area, the frame's beginning x and y coordinates are relative to the upper-left edge of the Page object.
5. Click **OK** to apply the changes and return to Template Builder.

To set a frame border:

You can add a customized border to each frame on the page.

1. On the **Frame** menu, click **Border**. The Frame Border dialog appears.
2. From the list, select the frame you want to add a border to.
3. In the **Border Type** section, enable the **Plain** option to add a border to the selected field.
4. In the **Border Width** field, enter the border width.
5. In the **Border Color** section, choose a border color either by using the three RGB scrollbars or by typing a color name.
6. Click **OK** to apply the changes and return to Template Builder.

To add columns to frame:

1. On the **Frame** menu, click **Columns**. the Frame Columns dialog appears.
2. From the list, select the frame you want to add columns to.
3. In the **Column Count** field, enter the number of frame columns you want.
4. In the **Column Spacing** field, enter the space between each column.
5. Click **OK** to apply the changes and return to Template Builder.

11.2.3 Step 3: Setting Text Flow on the Page

After creating your frames, the next step is to set the frame flow. This determines how text on the page moves from frame to frame.

To set the text flow:

1. On the **Flow** menu, click **Frame list**. The Frame List dialog appears.

   The Frames list displays all the frames on the current page. The Current Frame Flow list displays the current order (if any).
2. To add a frame to the Current Frame Flow list, select it and click **Add**. Repeat this process for every frame on the page.
3. To change the frame order in the Current Frame Flow list, select it and use the **Up** and **Down** buttons.

11.2.4 Step 4: Create other pages

Repeat steps 1 through 4 for each page you want to add to the document template.

11.2.5 Step 5: Setting Page Flow

After you have created all the pages and frames in your document, the final step is to connect the pages by setting the *page flow*. The page flow determines how text streams between pages.

Use the following procedures to set form feeds and section breaks. A form feed sends text to the next page in the flow, and occurs whenever *KlgPfFlowFormFeed()* is called. A new Page object is created, based on the `FlowPageTemplate` attribute in the template file.

A section break is useful for signalling the beginning of a new chapter or section, and occurs whenever *KlgPfFlowSectionBreak()* is called. A new Page object is created based on the `FlowSectionTemplate` attribute.

For more information on *KlgPfFlowFormFeed()* and *KlgPfFlowSectionBreak(),* see the Document Object on page 147.

To set the page flow:

1. Open the page that you want to set the page flow for.
2. On the **Flow** menu, click **Flow Page**. The Flow Page dialog appears.
3. In the list of template pages, select the next page you want text to flow to.
4. Click **OK** to save the change and return to Template Builder.

To set a section break:

1. Open the page that you want to set the section break for.
2. On the **Flow** menu, click **Section Page**. The Section Page dialog appears.
3. In the list of template pages, select the page you want to use to start a new section.
4. Click **OK** to save the change and return to Template Builder.

11.2.6 Step 6: Saving Your Template

After you have created your document template, save it for use in your code.

To save your template:

1. On the **File** menu, click **Save**. The Save dialog appears.
2. Type the file name you want to use and click **OK**.

Part II

Reference Appendices

A

Color Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Color object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

A.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfColor.h>

Parent Object: Frame Object, Page Object, Style Object

Usage: Frame object: `BorderColor`, `Color`
Page object: `Color`
Style object: `Color`

Description: The Color object provides methods for the creation of colors. The Color object creates a color by specifying the intensity of `Red`, `Blue`, and `Green` attributes. **Note:** You must destroy all Color objects you create.

A.2 Attributes (Name, Type, Default)

Blue **double** **0.0**

The intensity of blue, expressed as a value between 0 and 1 (0 being no blue and 1 being blue at its maximum intensity).

Green **double** **0.0**

The intensity of green, expressed as a value between 0 and 1 (0 being no green and 1 being green at its maximum intensity).

Red **double** **0.0**

The intensity of red, expressed as a value between 0 and 1.0 (0 being no red and 1.0 being red at its maximum intensity).

A.3 Methods

KlgPfColorCopy()

Copies a Color object.

```
KlgPfObject
KlgPfColorCopy(
   KlgPfObject   color
   )
```

Returns

Returns a copy of the Color object.

Parameters

color: The Color object you wish to duplicate.

KlgPfColorCreate()

Creates a new Color object. The method initializes `Red`, `Green`, and `Blue` attributes to 0.

```
KlgPfObject
KlgPfColorCreate(
   void
   )
```

Returns

Returns a Color object.

KlgPfColorCreateFromString()

Use this method to create a Color object and set `Red`, `Green`, and `Blue` attributes by using one of the predefined color strings listed below. *KlgPfColorCreateFromString()* is not case sensitive. Any string with "gray" may also be passed as "grey" (for example, "Grey62" is equivalent to "gray62").

```
KlgPfObject
KlgPfColorCreateFromString(
   String        color_str
   )
```

Returns

Returns a Color object.

Parameters

color_str: A string corresponding to one of the following color names:

"AliceBlue", "AntiqueWhite", "Aquamarine", "Azure", "Beige", "Bisque", "Black", "BlanchedAlmond", "Blue", "BlueViolet", "Brown", "Burlywood", "CadetBlue", "Chartreuse", "Chocolate", "Coral", "CornflowerBlue", "Cornsilk", "Cyan",

"DarkGoldenrod", "DarkGreen", "DarkGrey", "DarkKhaki", "DarkOliveGreen", "DarkOrange", "DarkOrchid", "DarkSalmon", "DarkSeaGreen", "DarkSlateBlue", "DarkSlateGray", "DarkTurquoise", "DarkViolet", "DeepPink", "DeepSkyBlue", "DodgerBlue", "Firebrick", "FloralWhite", "ForestGreen", "Gainsboro", "GhostWhite", "Gold", "Goldenrod", "Gray", "Green", "GreenYellow", "Honeydew", "HotPink", "IndianRed", "Ivory", "Khaki", "Lavender", "LavenderBlush", "LawnGreen", "LemonChiffon", "LightBlue", "LightCoral", "LightCyan", "LightGoldenrod", "LightGoldenrodYellow", "LightGray", "LightGrey", "LightPink", "LightSalmon", "LightSeaGreen", "LightSkyBlue", "LightSlateBlue", "LightSlateGray", "LightSteelBlue", "LightYellow", "LimeGreen", "Linen", "Magenta", "Maroon", "MediumAquamarine", "MediumBlue", "MediumOrchid", "MediumPurple", "MediumSeaGreen", "MediumSlateBlue", "MediumSpringGreen", "MediumTurquoise", "MediumVioletRed", "MidnightBlue", "MintCream", "MistyRose", "Moccasin", "NavajoWhite", "NavyBlue", "OldLace", "OliveDrab", "Orange", "OrangeRed", "Orchid", "PaleGoldenrod", "PaleGreen", "PaleTurquoise", "PaleVioletRed", "PapayaWhip", "PeachPuff", "Peru", "Pink", "Plum", "PowderBlue", "Purple", "Red", "RosyBrown", "RoyalBlue", "SaddleBrown", "Salmon", "SandyBrown", "SeaGreen", "Seashell", "Sienna", "SkyBlue", "SlateBlue", "SlateGray", "Snow", "SpringGreen", "SteelBlue", "Tan", "Thistle", "Tomato", "Turquoise", "Violet", "VioletRed", "Wheat", "White", "Yellow", "YellowGreen", "Gray1", "Gray2", "Gray3", "Gray4", "Gray5", "Gray6", "Gray7", "Gray8", "Gray9", "Gray10", "Gray11", "Gray12", "Gray13", "Gray14", "Gray15", "Gray16", "Gray17", "Gray18", "Gray19", "Gray20", "Gray21", "Gray22", "Gray23", "Gray24", "Gray25", "Gray26", "Gray27", "Gray28", "Gray29", "Gray30", "Gray31", "Gray32", "Gray33", "Gray34", "Gray35", "Gray36", "Gray37", "Gray38", "Gray39", "Gray40", "Gray41", "Gray42", "Gray43", "Gray44", "Gray45", "Gray46", "Gray47", "Gray48", "Gray49", "Gray50", "Gray51", "Gray52", "Gray53", "Gray54", "Gray55", "Gray56", "Gray57", "Gray58", "Gray59", "Gray60", "Gray61", "Gray62", "Gray63", "Gray64", "Gray65", "Gray66", "Gray67", "Gray68", "Gray69", "Gray70", "Gray71", "Gray72", "Gray73", "Gray74", "Gray75", "Gray76", "Gray77", "Gray78", "Gray79", "Gray80", "Gray81", "Gray82", "Gray83", "Gray84", "Gray85", "Gray86", "Gray87", "Gray88", "Gray89", "Gray90", "Gray91", "Gray92", "Gray93", "Gray94", "Gray95", "Gray96", "Gray97", "Gray98", and "Gray99".

KlgPfColorCreateGrey()

Creates a Color object with a given grey level.

```
KlgPfObject
KlgPfColorCreateGrey(
   double        grey_level
   )
```

Returns

Returns a Color object.

Parameters

grey_level: Must be a value between 0 and 1. 0 is black. 1 is white. 0.5 is 50% grey. The method sets the `Red`, `Green`, and `Blue` attributes to the value specified by *grey_level.*

KlgPfColorCreateRGB()

Creates a Color object with the given set of *red*, *blue*, and *green* intensities.

```
KlgPfObject
KlgPfColorCreateRGB(
   double        red,
   double        green,
   double        blue
   )
```

Returns

Returns a Color object.

Parameters

red/green/blue: Must be a value between 0 and 1.0.

KlgPfColorDestroy()

Destroys a Color object. Note: destroying the parent object automatically destroys this object. You should not call this method if you have destroyed the parent object.

```
void
KlgPfColorDestroy(
   KlgPfObject   color_object
   )
```

Parameters

color: The Color object you wish to destroy.

KlgPfColorGetBlue()

Retrieves the value of the Color object's `Blue` attribute.

```
double
KlgPfColorGetBlue(
   KlgPfObject   color
   )
```

KlgPfColorGetGreen()

Retrieves the value of the Color object's `Green` attribute.

```
double
KlgPfColorGetGreen(
   KlgPfObject   color
   )
```

KlgPfColorGetGrey()

Retrieves the Color object's grey level.

```
double
KlgPfColorGetGrey(
   KlgPfObject   color
   )
```

Returns

Returns the value of the Color object's `red` attribute. This method only returns the value of the `Red` attribute since `Red`, `Green`, and `Blue` are set to the same value when assigning grey levels with *KlgPfColorSetGrey()* and *KlgPfColorCreateGrey()*.

Parameters

color: A Color object.

KlgPfColorGetRed()

Retrieves the value of the Color object's `Red` attribute.

```
double
KlgPfColorGetRed(
   KlgPfObject   color
   )
```

KlgPfColorGetRGB()

Returns the Color object's `Red`, `Green`, and `Blue` attributes.

```
void
KlgPfColorGetRGB(
   KlgPfObject   color,
   double        *red,      /* Returned */
   double        *green,    /* Returned */
   double        *blue      /* Returned */
   )
```

Parameters

color: A Color object.

red/green/blue: Pointers to variables of type double for the color attributes.

KlgPfColorSetBlue()

Sets the Color object's `Blue` attribute.

```
Boolean
KlgPfColorSetBlue(
   KlgPfObject   color,
   double        value
   )
```

Returns

Returns `True` if the attribute is set successfully.

Parameters

color: A Color object.

value: A value between 0 and 1.

KlgPfColorSetGreen()

Sets the Color object's `Green` attribute.

```
Boolean
KlgPfColorSetGreen(
   KlgPfObject   color,
   double        value
   )
```

Returns

Returns `True` if the attribute is set successfully.

Parameters

color: A Color object.

value: A value between 0 and 1.

KlgPfColorSetGrey()

Specifies a grey level by setting the Color object's `Red`, `Green`, and `Blue` attributes to the same value.

```
Boolean
KlgPfColorSetGrey(
   KlgPfObject   color,
   double        value
   )
```

Returns

Returns `True` if the attribute is set successfully.

Parameters

color: A Color object.

value: A value between 0 and 1.

KlgPfColorSetRed()

Sets the Color object's `Red` attribute.

```
Boolean
KlgPfColorSetRed(
   KlgPfObject   color,
   double        value
   )
```

Returns

Returns `True` if the attribute is set successfully.

Parameters

color: A Color object.

value: A value between 0 and 1.

KlgPfColorSetRGB()

Sets the Color object's `Red`, `Green`, and `Blue` attributes.

```
Boolean
KlgPfColorSetRGB(
   KlgPfObject   color,
   double        red,
   double        green,
   double        blue
   )
```

Returns

Returns `True` if the attribute is set successfully.

Parameters

color: A Color object.

red/green/blue: Must be a value between 0 and 1.

KlgPfColorTestEqual()

Tests if all components of one Color object are equal to the corresponding components of another.

```
Boolean
KlgPfColorTestEqual(
   KlgPfObject   color1,
   KlgPfObject   color2
   )
```

Returns

Returns `True` if the color objects are equal.

Parameters

color1/color2: Color objects.

KlgPfObjectIsColor()

Checks if the object is a Color object.

```
Boolean
KlgPfObjectIsColor(
   KlgPfObject   object
   )
```

Returns

Returns `True` if the object is a Color object.

Parameters

object: An object.

B

Display X11 Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Display X11 object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

B.1 Object Synopsis

Include File: <KlgPfDisplayX11.h>

Parent Object: Printer Object

Usage: X Windows Printer: `Display`

Description: The Display X11 object holds X11 server information that allows a printer of type `KLGPF_PRINTER_TYPE_X11` to display a document on a screen. The printer uses attributes from the given `Display/Window` combination to render a document to the specified `Drawable` (normally the same as the `Window` attribute).

B.2 Attributes (Name, Type, Default)

Display **Display *** ***Dynamic***

Specifies a pointer to X11 display on which the document is to be printed.

Window **Window** ***Dynamic***

Specifies a window whose attributes are used to render the document to the drawable.

Drawable **Drawable** ***Dynamic***

Specifies the drawable to which the document is rendered. By default, this is equal to the value of the `Window` attribute. It can be set to any drawable (for example, a pixmap) that was created from the given `Display/Window` combination.

B.3 Methods

KlgPfDisplayX11Create()

Creates and returns a Display X11 object. PageFormatter sets `Drawable` to the same value as `Window`.

```
KlgPfObject
KlgPfDisplayX11Create(
   Display       *display,
   Window        window
   )
```

Parameters

display: A pointer to a display.

window: A window.

KlgPfDisplayX11Destroy()

Destroys a Display X11 object.

```
void
KlgPfDisplayX11Destroy(
   KlgPfObject   displayx11_object
   )
```

KlgPfDisplayX11GetDisplay()

Returns a pointer to the Display X11 object's `Display`.

```
Display *
KlgPfDisplayX11GetDisplay(
   KlgPfObject   displayx11_object
   )
```

KlgPfDisplayX11GetDrawable()

Returns the Display X11 object's current drawable pixmap.

```
Drawable
KlgPfDisplayX11GetDrawable(
   KlgPfObject   displayx11_object
   )
```

KlgPfDisplayX11GetWindow()

Returns the Display X11 object's `Window`.

```
Window
KlgPfDisplayX11GetWindow(
   KlgPfObject   displayx11_object
   )
```

KlgPfDisplayX11SetDisplay()

Sets the Display X11 object's `Display`.

```
Boolean
KlgPfDisplayX11SetDisplay(
   KlgPfObject   displayx11_object,
   Display       *display
   )
```

Parameters

displayX11_obj: The Display X11 object.

display: A pointer to the display.

KlgPfDisplayX11SetDrawable()

Sets the Display X11 object's `Drawable`.

```
Boolean
KlgPfDisplayX11SetDrawable(
   KlgPfObject   displayx11_object,
   Drawable      drawable
   )
```

KlgPfDisplayX11SetWindow()

Sets the Display X11 object's `Window`.

```
Boolean
KlgPfDisplayX11SetWindow(
   KlgPfObject   displayx11_object,
   Window        window
   )
```

KlgPfObjectIsDisplayX11()

Returns `True` if the object is a Display X11 object.

```
Boolean
KlgPfObjectIsDisplayX11(
   KlgPfObject   object
   )
```

C

Document Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Document object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

C.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfDocument.h>

Description: The Document object provides attributes and methods for controlling the flow through the rendering process.

C.2 Attributes (Name, Type, Default)

CurrentFlow **int** **0**

The index number of the current frame in the flow frame list.

CurrentFrame **KlgPfObject** **NULL**

The current Frame object to which the Document object is rendering.

CurrentPage **KlgPfObject** **NULL**

The current Page object to which the Document object is rendering.

CurrentStyle **KlgPfObject** **NULL**

The current Style object the Document is applying to the rendered text.

CurrentDrawStyle **KlgPfObject** **NULL**

The current DrawStyle object the Document is applying to rendered lines.

DrawStyleList **KlgPfObject** ***Dynamic***

A list of all DrawStyle objects available to the Document object.

FlushPolicy **KlgPfFlushPolicy** **KLGPF_FLUSH_NEVER**

Specifies when a Page object should be destroyed. Valid values are:

`KLGPF_FLUSH_NEVER`	The Page objects will not be destroyed until the Document object is destroyed.
`KLGPF_FLUSH_ON_OUTPUT`	A Page object is destroyed after it's been printed.

FrameCallback **KlgPfCallbackRecord** **NULL**

Specifies a procedure (along with a pointer to any user-defined data) that should be called when a frame callback is generated. When PageFormatter generates a frame callback, it passes the following structure to the callback procedure:

```
typedef struct {
   KlgPfCallbackReason  reason;      /* Read-only */
   KlgPfObject          frame;       /* Read-only */
   KlgPfObject          document;    /* Read-only */
   KlgPfObject          printer;     /* Read-only */
} KlgPfFrameCallbackStruct;
```

The *reason* member is one of the following:

`KLGPF_REASON_FRAME_NEW`	A new frame has been generated.
`KLGPF_REASON_FRAME_FLOW`	The flow is about to be transferred to a new frame. This callback is generated *before* text is rendered to the new frame.
`KLGPF_REASON_FRAME_END`	The frame has been completely rendered.

Note: the `KlgPfCallbackRecord` structure is settable (via *KlgPfDocSetFrameCallback()*) but not gettable.

MemoryMax **int** **0**

Specifies the maximum amount of memory that should be used for the rendering process. If memory use exceeds `MemoryMax`, PageFormatter will write rendered pages to a temporary file to keep page memory usage below `MemoryMax`. If `MemoryMax` is set to `0`, PageFormatter will not use a temporary file.

Name **String** ***Dynamic***

ASCII name of the Document object.

OutputPolicy **KlgPfOutputPolicy** **KLGPF_OUTPUT_AS_REQUESTED**

Specifies when rendered pages should be physically printed. Valid values are:

KLGPF_OUTPUT_AS_REQUESTED	Specifies pages should be output when the document has been completely rendered.
KLGPF_OUTPUT_ASAP	Specifies each page should be output as soon as it is completely rendered.

PageCallback **KlgPfCallbackRecord** **NULL**

Specifies a procedure (along with a pointer to any user-defined data) that should be called when a page callback is generated. When PageFormatter generates a page callback, it passes the following structure to the callback procedure:

```
typedef struct {
   KlgPfCallbackReason    reason;      /* Read-only */
   KlgPfObject            page;        /* Read-only */
   KlgPfObject            document;    /* Read-only */
   KlgPfObject            printer;     /* Read-only */
} KlgPfPageCallbackStruct;
```

The *reason* member is one of the following enumerated types:

KLGPF_REASON_PAGE_NEW	A new page has been generated.
KLGPF_REASON_PAGE_FLOW	The flow is about to be transferred to a new page. This callback is generated *before* text is rendered to the first frame on the new page.
KLGPF_REASON_PAGE_END	The page has been completely rendered.

Note: the KlgPfCallbackRecord structure is settable (via *KlgPfDocSetPageCallback()*) but not getable.

PageCount **int** **0**

The total number of pages currently rendered. This attribute is increased by 1 each time a page is created. *No set function is provided for this attribute.*

PageList **KlgPfObject** ***Dynamic***

A list of already rendered or partially rendered Page objects.

PageNumber **int** **0**

The page number of the page currently being rendered. This attribute is increased by 1 each time a page is created. Unlike PageCount, this attribute can be changed with *KlgPfDocSetPageNumber()*.

PageTemplateList **KlgPfObject** ***Dynamic***

A list of all page templates available to the Document object.

Printer **KlgPfObject** ***Dynamic***

Specifies the Printer object for which the document is being rendered.

SectionCallback **KlgPfCallbackRecord** **NULL**

Specifies a procedure (along with a pointer to any user-defined data) that should be called when a section callback is generated. When PageFormatter generates a section callback, it passes the following structure to the callback procedure:

```
typedef struct {
   KlgPfCallbackReason   reason;       /*Read-only */
   KlgPfObject           document;     /*Read-only */
   KlgPfObject           printer;      /*Read-only */
} KlgPfSectionCallbackStruct;
```

The *reason* member is one of the following values:

KLGPF_REASON_SECTION_NEW	A new page has been generated because of a *KlgPfFlowSectionBreak()* call. This callback happens *before* a KLGPF_REASON_PAGE_NEW callback is generated.
KLGPF_REASON_SECTION_FLOW	The flow is about to be transferred because of a *KlgPfFlowSectionBreak()* call.

Note: the KlgPfCallbackRecord structure is setable (via *KlgPfDocSetSectionCallback()*) but not getable.

StyleList **KlgPfObject** ***Dynamic***

A list of all Style objects available to the Document object.

TableList **KlgPfObject** ***Dynamic***

A list of all Table objects available to the Document object.

Unit **KlgPfUnit** **Inches**

The Document object's dimension units or document units. Many methods take height, width, and (x, y) point parameters. These parameters are interpreted by PageFormatter based on this attribute. Valid values are:

KLGPF_UNIT_CM	Centimeters
KLGPF_UNIT_INCHES	Inches
KLGPF_UNIT_POINTS	Points

C.3 Methods

KlgPfDocBegin()

Initializes the first page to be rendered based on the first template in the Document object's `PageTemplateList` and renders the background objects for the page and its frames.

```
Boolean
KlgPfDocBegin(
   KlgPfObject       document
   )
```

Returns

Returns `True` if the Document object is successfully created.

Parameters

document: The Document object to be rendered to.

KlgPfDocCreate()

Creates and returns a Document object.

```
KlgPfObject
KlgPfDocCreate(
   KlgPfObject       printer,
   String            name
   )
```

Parameters

printer: A Printer object (which determines how to render any data flowed to the Document object).

name: The name string of the Document object (for example, "mydocument").

KlgPfDocDestroy()

Destroys a Document object. This method destroys all child objects (Styles, Pages, and so on). You do not have to destroy individual child objects.

```
void
KlgPfDocDestroy(
   KlgPfObject       document
   )
```

KlgPfDocEnd()

Ends the rendering process for this document. Any pages or frames that still have unresolved macros (for example, total page macros) will be completed.

```
Boolean
KlgPfDocEnd(
   KlgPfObject       document
   )
```

KlgPfDocGetCurrentFlow()

Gets the index number of the current frame in the Document object's `FlowFrameList`.

```
int
KlgPfDocGetCurrentFlow(
   KlgPfObject       document
   )
```

Returns

Returns the value of the Document object's `CurrentFlow` attribute.

KlgPfDocGetCurrentDrawStyle()

Retrieves the current DrawStyle object PageFormatter is currently using to render lines and fills via the Frame object's KlgPfDraw methods.

```
KlgPfObject
KlgPfDocGetCurrentDrawStyle(
   KlgPfObject       document
   )
```

KlgPfDocGetCurrentFrame()

Retrieves the Frame object to which PageFormatter is currently rendering.

```
KlgPfObject
KlgPfDocGetCurrentFrame(
   KlgPfObject       document
   )
```

KlgPfDocGetCurrentPage()

Retrieves the Page object to which PageFormatter is currently rendering.

```
KlgPfObject
KlgPfDocGetCurrentPage(
   KlgPfObject       document
   )
```

KlgPfDocGetCurrentStyle()

Retrieves the Style object PageFormatter is using to currently render text.

```
KlgPfObject
KlgPfDocGetCurrentStyle(
   KlgPfObject       document
   )
```

KlgPfDocGetDrawStyleList()

Returns the list of all DrawStyle objects available to the Document object.

```
KlgPfObject
KlgPfDocGetDrawStyleList(
   KlgPfObject       document
   )
```

KlgPfDocGetFlushPolicy()

Retrieves the Flush Policy (that is the policy used to determine when PageFormatter should destroy Page objects).

```
KlgPfFlushPolicy
KlgPfDocGetFlushPolicy(
   KlgPfObject       document
   )
```

KlgPfDocGetMemoryMax()

Gets the maximum amount of memory PageFormatter is allowed for storage of rendered pages.

```
int
KlgPfDocGetMemoryMax(
   KlgPfObject       document
   )
```

KlgPfDocGetName()

Retrieves the value of the Document object's `Name` attribute.

```
String
KlgPfDocGetName(
   KlgPfObject       document
   )
```

KlgPfDocGetOutputPolicy()

Gets the output policy which defines when rendered pages should be written to the current output device.

```
KlgPfOutputPolicy
KlgPfDocGetOutputPolicy(
   KlgPfObject       document
   )
```

KlgPfDocGetPageCount()

Retrieves the total number of pages currently rendered.

```
int
KlgPfDocGetPageCount(
   KlgPfObject       document
   )
```

KlgPfDocGetPageList()

Retrieves the list of all rendered Page objects.

```
KlgPfObject
KlgPfDocGetPageList(
   KlgPfObject       document
   )
```

KlgPfDocGetPageNumber()

Returns the page number of the page currently being rendered.

```
int
KlgPfDocGetPageNumber(
   KlgPfObject       document
   )
```

KlgPfDocGetPageTemplateList()

Returns the list of all template Page objects available to the Document object.

```
KlgPfObject
KlgPfDocGetPageTemplateList(
   KlgPfObject       document
   )
```

KlgPfDocGetPrinter()

Returns the Document object's specified Printer object.

```
KlgPfObject
KlgPfDocGetPrinter(
   KlgPfObject       document
   )
```

KlgPfDocGetStyleList()

Returns the list of all Style objects available to the Document object. When you create a Style object, it automatically gets added to this list.

```
KlgPfObject
KlgPfDocGetStyleList(
   KlgPfObject       document
   )
```

KlgPfDocGetUnit()

Returns the Document's dimension units.

```
KlgPfUnit
KlgPfDocGetUnit(
   KlgPfObject       document
   )
```

KlgPfDocSetCurrentDrawStyle()

Sets the Draw Style object to be applied to a subsequent render action.

```
Boolean
KlgPfDocSetCurrentDrawStyle(
   KlgPfObject       document
   KlgPfObject       style
   )
```

KlgPfDocSetCurrentFlow()

Sets which frame in the flow frame list to which the Document object should render.

```
Boolean
KlgPfDocSetCurrentFlow(
   KlgPfObject       document,
   int               flow_index
   )
```

Parameters

document: A Document object.

flow_index: The index number of the desired frame (0 based).

KlgPfDocSetCurrentStyle()

Sets the Style object to be applied to text in subsequent render actions.

```
Boolean
KlgPfDocSetCurrentStyle(
   KlgPfObject       document,
   KlgPfObject       style
```

KlgPfDocSetFlushPolicy()

Sets the Flush Policy (`KLGPF_FLUSH_NEVER` or `KLGPF_FLUSH_ON_OUTPUT`) PageFormatter uses to determine when Page objects should be destroyed.

```
Boolean
KlgPfDocSetFlushPolicy(
   KlgPfObject       document,
   KlgPfFlushPolicy  policy
   )
```

KlgPfDocSetFrameCallback()

Directs PageFormatter to call the supplied procedure (along with a handle for any user-defined data) when PageFormatter changes frames.

```
Boolean
KlgPfDocSetFrameCallback(
   KlgPfObject       document,
   KlgPfCallbackProc callback,
   XtPointer         call_data
   )
```

Parameters

document: The Document object.

callback: The procedure to call. The procedure must be defined to be the following type:

```
typedef void (*KlgPfCallbackProc)(
   XtPointer         cbs,
   XtPointer         call_data
   );
```

call_data: A user-created data handle.

KlgPfDocSetMemoryMax()

Sets the maximum amount of memory (in bytes) that should be used for storing rendered pages.

```
Boolean
KlgPfDocSetMemoryMax(
   KlgPfObject       document,
   int               size
   )
```

KlgPfDocSetOutputPolicy()

Defines when rendered pages should be written. Note: PageFormatter will change `OutputPolicy` back to `KLGPF_OUTPUT_AS_REQUESTED` automatically if it encounters a page with macros that can't be resolved until all pages are completely rendered.

```
Boolean
KlgPfDocSetOutputPolicy(
   KlgPfObject       document,
   KlgPfOutputPolicy policy
   )
```

Parameters

document: The Document object.

policy: An output policy type (for example, `KLGPF_OUTPUT_AS_REQUESTED`).

KlgPfDocSetPageCallback()

Directs PageFormatter to call the supplied procedure (along with a handle for any user-defined data) when PageFormatter changes pages.

```
Boolean
KlgPfDocSetPageCallback(
   KlgPfObject       document,
   KlgPfCallbackProc callback,
   XtPointer         call_data
   )
```

Parameters

document: The Document object.

callback: The procedure to call. The procedure must be defined to be the following type:

```
typedef void (*KlgPfCallbackProc)(
   XtPointer         cbs,
   XtPointer         call_data
   );
```

call_data: A user-created data handle.

KlgPfDocSetPageNumber()

Sets the page number of the page currently being rendered.

```
Boolean
KlgPfDocSetPageNumber(
   KlgPfObject       document,
   int               page_number
```

KlgPfDocSetPrinter()

Sets which Printer object the Document should use for rendering purposes.

```
Boolean
KlgPfDocSetPrinter(
   KlgPfObject       document,
   KlgPfObject       printer
   )
```

KlgPfDocSetSectionCallback()

Directs PageFormatter to call the supplied procedure (along with a handle for any user-defined data) when PageFormatter changes sections.

```
Boolean
KlgPfDocSetSectionCallback(
   KlgPfObject       document,
   KlgPfCallbackProc callback,
   XtPointer         call_data
   )
```

Parameters

document: The Document object.

callback: The procedure to call. The procedure must be defined to be the following type:

```
typedef void (*KlgPfCallbackProc)(
   XtPointer         cbs,
   XtPointer         call_data
   );
```

call_data: A user-created data handle.

KlgPfDocSetUnit()

Sets the Document's dimension units. Setting the Document object's `Unit` attribute automatically sets the Page and Frame objects' `Unit` attributes. *Unit* can be inches (`KLGPF_UNIT_INCHES`), centimeters (`KLGPF_UNIT_CM`), or points (`KLGPF_UNIT_POINTS`).

```
Boolean
KlgPfDocSetUnit(
   KlgPfObject       document,
   KlgPfUnit         unit
   )
```

KlgPfDocStringToDrawStyle()

Returns the Draw Style object with the given name (for example, "double_rule") or `NULL` if the named style is not found. *style_name* does not have to match the case of the style name.

```
KlgPfObject
KlgPfDocStringToDrawStyle(
   KlgPfObject       document,
   String            style_name
   )
```

KlgPfDocStringToStyle()

Returns the Style object with the given name (for example, "heading1") or `NULL` if the named style is not found. *style_name* does not have to match the case of the style name.

```
KlgPfObject
KlgPfDocStringToStyle(
   KlgPfObject       document,
   String            style_name
   )
```

KlgPfDocStringToTemplate()

Gets a template Page object by name (for example, "TitlePage") from the Document object's `PageTemplateList`. Returns the template or `NULL` if not found. PageFormatter is not case sensitive.

```
KlgPfObject
KlgPfDocStringToTemplate(
   KlgPfObject       document,
   String            template_name
   )
```

KlgPfFlowColumnFeed()

Advances the flow to the top of the next column. If there is no next column, *KlgPfFlowFrameFeed()* is called.

```
Boolean
KlgPfFlowColumnFeed(
   KlgPfObject       document
   )
```

KlgPfFlowFormFeed()

Generates a new page based on the template Page object defined by the current Page object's `FlowPageTemplate` attribute. The flow is transferred to the first frame on the new page.

```
Boolean
KlgPfFlowFormFeed(
   KlgPfObject       document
   )
```

Returns

Returns `True` if the flow is transferred successfully.

KlgPfFlowFrameFeed()

Generates a frame feed and transfers the flow to the next frame indicated by the current Page object's `FlowFrameList` attribute. If there is no next frame, a *KlgPfFlowFormFeed()* is called.

```
Boolean
KlgPfFlowFrameFeed(
   KlgPfObject       document
   )
```

Returns

Returns `True` if the flow is transferred successfully.

KlgPfFlowHRuleLength()

Draws a horizontal ruling line in the flow, of a specified length. The line's vertical width and line color are defined by the Document's `CurrentDrawStyle` attribute.

```
Boolean
KlgPfFlowHRuleLength(
   KlgPfObject       document,
   double            length
   )
```

Parameters

document: The Document object.

length: The rule's length, in document units.

KlgPfFlowHRulePercent()

Draws a horizontal ruling line in the flow. The line's length is expressed as a percentage of the current column's width. The line's vertical width and line color are defined by the Document's `CurrentDrawStyle` attribute.

```
Boolean
KlgPfFlowHRulePercent(
   KlgPfObject       document,
   double            percent
   )
```

Parameters

document: The Document object.

percentage: The rule's length, expressed as a percentage of the current column's width.

KlgPfFlowImageEmbed()

Draws the specified Image object as an inline graphic.

```
Boolean
KlgPfFlowImageEmbed(
   KlgPfObject         document,
   KlgPfObject         image,
   KlgPfImageAlignment alignment
   )
```

Parameters

document: The Document object.

image: An Image object.

alignment: An alignment value. Images may be positioned, relative to the text in the line, according to one of the following alignments:

KLGPF_IMAGE_BASELINE	Aligns the image at the text's baseline.
KLGPF_IMAGE_BOTTOM	Aligns the image at the bottom of the line.
KLGPF_IMAGE_CENTER	Vertically centers the image in the line.
KLGPF_IMAGE_TOP	Aligns the image at the top of the line.

KlgPfFlowImageFloat()

Draws the specified Image object on its own line. If there is not enough room on the page to render the object, PageFormatter will hold the image and continue rendering data in the flow. *KlgPfFlowImageFloat()* will lay out the image when a new column or frame having sufficient space is generated. If sufficient space is not found before *KlgPfDocEnd()*, the image will not be rendered.

```
Boolean
KlgPfFlowImageFloat(
   KlgPfObject       document,
   KlgPfObject       image
   )
```

Parameters

document: The Document object.

image: An Image object.

KlgPfFlowLineFeed()

Generates a line feed, staring a new paragraph, and transfers the flow to the next line.

```
Boolean
KlgPfFlowLineFeed(
   KlgPfObject       document
   )
```

Returns

Returns `True` if the flow is transferred successfully.

KlgPfFlowPrint()

Renders the specified string to the current flow frame. If all or part of the text does not fit in this frame, PageFormatter will automatically print the remaining text in the next frame.

```
Boolean
KlgPfFlowPrint(
   KlgPfObject       document,
   String            text
   )
```

Returns

Returns `True` if the flow is rendered successfully.

Parameters

document: The Document object.

text: A string of text to be rendered.

KlgPfFlowSectionBreak()

Generates a section break. *KlgPfFlowSectionBreak()* assigns a new page template based on the Page object's `FlowSectionTemplate` attribute, generates a new page, and transfers the flow to the first frame on the new page.

```
Boolean
KlgPfFlowSectionBreak(
   KlgPfObject       document
   )
```

Returns

Returns `True` if the section break is successful.

KlgPfFlowTab()

Generates a tab feed in the current flow. *KlgPfFlowTab()* uses the next Tab object in the current Style object's `TabList` with an `X` attribute greater than the current position when the method was called.

```
Boolean
KlgPfFlowTab(
   KlgPfObject       document
   )
```

Returns

Returns `True` if the tab feed is successful.

KlgPfObjectIsDoc()

Verifies that the object is a Document object.

```
Boolean
KlgPfObjectIsDoc(
   KlgPfObject       object
   )
```

Returns

Returns `True` if the object is a Document object.

D

DrawStyle Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the DrawStyle object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

D.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfDrawStyle.h>

Parent Object: Document Object

Usage: `CurrentDrawStyle`, `DrawStyleList`

Description: The DrawStyle object provides methods for the creation of styles used by the Frame object's drawing methods (for example, *KlgPfDrawSimpleLine()*).

D.2 Attributes (Name, Type, Default)

Color **KlgPfObject** **"Black"**

A Color object specifying the line color and fill color.

FillRule **KlgPfFillRule** **KLGPF_FILL_RULE_NON_ZERO_WINDING**

The `FillRule` attribute specifies the rule used to determine the interior of a filled area created by the *KlgPfDrawFilledPolyline()* drawing method. Valid values are:

KLGPF_FILL_RULE_EVEN_ODD	Defines a point to be inside if any infinite ray starting at the point crosses the polyline an odd number of times.
KLGPF_FILL_RULE_NON_ZERO_WINDING	Defines a point to be inside if any infinite ray starting at the point crosses an unequal number of clockwise or counterclockwise segments.

LineWidth **double** **0**

The line width in document units.

Name **String** ***Dynamic***

The drawing style's name.

D.3 Methods

KlgPfDrawStyleCopy()

Returns a copy of the DrawStyle Object.

```
KlgPfObject
KlgPfDrawStyleCopy(
   KlgPfObject   source,
   KlgPfObject   parent,
   String        name
   )
```

Parameters

source: The DrawStyle object you wish to copy.

parent: The Document object.

name: The new DrawStyle object's name.

KlgPfDrawStyleCreate()

Creates and returns a DrawStyle object.

```
KlgPfObject
KlgPfDrawStyleCreate(
   KlgPfObject   parent,
   String        name
   )
```

Parameters

parent: The Document object.

name: The DrawStyle object's name.

KlgPfDrawStyleDestroy()

Destroys the DrawStyle object. Note: destroying the Document object automatically destroys this object. In the normal course of actions, you would not call this method.

```
void
KlgPfDrawStyleDestroy(
   KlgPfObject   draw_style
   )
```

Parameters

draw_style: A DrawStyle object.

KlgPfDrawStyleGetColor()

Retrieves the DrawStyle object's `Color` attribute. Returns a Color object.

```
KlgPfObject
KlgPfDrawStyleGetColor(
   KlgPfObject   draw_style
   )
```

KlgPfDrawStyleGetFillRule()

Retrieves the DrawStyle object's `FillRule` attribute.

```
KlgPfFillRule
KlgPfDrawStyleGetFillRule(
   KlgPfObject   draw_style
   )
```

KlgPfDrawStyleGetLineWidth()

Retrieves the DrawStyle object's `LineWidth` attribute.

```
double
KlgPfDrawStyleGetLineWidth(
   KlgPfObject   draw_style
   )
```

KlgPfDrawStyleGetName()

Retrieves the DrawStyle object's `Name` attribute.

```
String
KlgPfDrawStyleGetName(
   KlgPfObject   draw_style
   )
```

KlgPfDrawStyleSetColor()

Sets the DrawStyle object's `Color` attribute.

```
Boolean
KlgPfDrawStyleSetColor(
   KlgPfObject   draw_style,
   KlgPfObject   color
   )
```

Parameters

draw_style: A DrawStyle object.

color: A Color object.

KlgPfDrawStyleSetFillRule()

Sets the DrawStyle object's `FillRule` attribute.

```
Boolean
KlgPfDrawStyleSetFillRule(
   KlgPfObject   draw_style,
   KlgPfFillRule fill_rule
   )
```

Parameters

draw_style: A DrawStyle object.

fill_rule: A `KlgPfFillRule` constant. Can be either `KLGPF_FILL_RULE_EVEN_ODD` or `KLGPF_FILL_RULE_NON_ZERO_WINDING`.

KlgPfDrawStyleSetLineWidth()

Sets the DrawStyle object's `LineWidth` attribute.

```
Boolean
KlgPfDrawStyleSetLineWidth(
   KlgPfObject   draw_style,
   double        width
   )
```

Parameters

draw_style: A DrawStyle object.

width: The line width in document units.

KlgPfObjectIsDrawStyle()

Returns `True` if the object is a DrawStyle object.

```
Boolean
KlgPfObjectIsDrawStyle(
   KlgPfObject   object
   )
```

Parameters

object: An object.

E

Font Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Font object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

E.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfFont.h>

Parent Object: Printer Object

Usage: Style object: `font`
Printer object: `FontList`

Description: The Font object is a child of the Printer object. PageFormatter represents every printer font available as a Font object. Font objects are held in the Printer object's `FontList` list attribute. When you create a Printer object, the `FontList` is generated automatically.

Note, a font should not be confused with a font family. A Font object represents one distinct printer font for a range of sizes (for example, "Times-Roman", "Times-Bold"). A font family represent a group of fonts using a similar typeface (for example, "Times") but representing different "weights" or "styles" (for example "Times-Roman", "Times-Bold", "Times-Italic").

E.2 Attributes (Name, Type, Default)

Name **String** ***Dynamic***

The name of the font (for example, "Times-Roman", "Times-Bold").

SupportedSizes **KlgPfObject** ***Dynamic***

Some fonts, such as screen fonts, are not scalable. If so, this attribute provides a list of supported sizes. If the list is `NULL`, the font is scalable.

E.3 Methods

KlgPfFontCreate()

Creates a Font object and adds itself to the indicated Printer object's `FontList` attribute.

```
KlgPfObject
KlgPfFontCreate(
   KlgPfObject   parent,
   String        name
   )
```

Returns

Returns a Font object.

Parameters

parent: The Font object's parent Printer object.

name: The name of the font (for example "Times-Roman").

KlgPfFontDestroy()

Destroys a Font object. Note: destroying the parent object automatically destroys this object. You should not call this method if you have destroyed the parent object.

```
void
KlgPfFontDestroy(
   KlgPfObject   font
   )
```

KlgPfFontGetName()

Gets the value of the Font object's `Name` attribute.

```
String
KlgPfFontGetName(
   KlgPfObject   font
   )
```

KlgPfFontGetSizesList()

Gets a list of the Font object's supported sizes.

```
KlgPfObject
KlgPfFontGetSupportedSizes(
   KlgPfObject   font
   )
```

KlgPfObjectIsFont()

Verifies the object is a Font object.

```
Boolean
KlgPfObjectIsFont(
   KlgPfObject   object
   )
```

Returns

Returns `True` if the object is a Font object.

F

Font Family Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Font Family object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

F.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfFontFamily.h>

Parent Object: Printer Object

Usage: Style object: `FontFamily`
Printer object: `FontFamilyList`

Description: The Font Family object provides methods and a structure for grouping together individual fonts into a "font family".

F.2 Attributes (Name, Type, Default)

Bold **KlgPfObject** **NULL**

The font to be used as the bold version of the font family.

BoldItalic **KlgPfObject** **NULL**

The font to be used as the bold-italic version of the font family.

Italic **KlgPfObject** **NULL**

The font to be used as the italic version of the font family.

Name **String** ***Dynamic***

The name of the Font Family (for example, “Times”, “Helvetica”).

Normal **KlgPfObject** **NULL**

The font to be used as the normal member (that is, the non-bolded, non-italicized member) of the font family.

F.3 Methods

KlgPfFontFamilyCreate()

Creates a Font Family object and adds it to the specified Printer object’s font family list.

```
KlgPfObject
KlgPfFontFamilyCreate(
   KlgPfObject   parent,
   String        name
   )
```

Returns

Returns a Font Family object.

Parameters

parent: The Font Family object’s parent Printer object

name: The name of the font family (for example “Times”).

KlgPfFontFamilyDestroy()

Destroys a Font Family object. Note: destroying the parent object automatically destroys this object. You should not call this method if you have destroyed the parent object.

```
void
KlgPfFontFamilyDestroy(
   KlgPfObject   font_family
   )
```

Parameters

font_family: The Font Family object.

KlgPfFontFamilyGetBold()

Returns the font to be used as the Font Family’s bold member.

```
KlgPfObject
KlgPfFontFamilyGetBold(
   KlgPfObject   font_family
   )
```

KlgPfFontFamilyGetBoldItalic()

Returns the font to be used as the Font Family's bold-italic member.

```
KlgPfObject
KlgPfFontFamilyGetBoldItalic(
   KlgPfObject   font_family
   )
```

KlgPfFontFamilyGetItalic()

Returns the font to be used as the Font Family's italic member.

```
KlgPfObject
KlgPfFontFamilyGetItalic(
   KlgPfObject   font_family
   )
```

KlgPfFontFamilyGetName()

Returns the Font Family's name.

```
String
KlgPfFontFamilyGetName(
   KlgPfObject   font_family
   )
```

KlgPfFontFamilyGetNormal()

Returns the font to be used as the Font Family's normal member.

```
KlgPfObject
KlgPfFontFamilyGetNormal(
   KlgPfObject   font_family
   )
```

KlgPfFontFamilySetBold()

Sets the Font object that should be used as the Font Family's bold member.

```
Boolean
KlgPfFontFamilySetBold(
   KlgPfObject   font_family,
   KlgPfObject   bold
   )
```

Parameters

font_family: The Font Family object.

bold: Font object to be used as the bold font.

KlgPfFontFamilySetBoldItalic()

Sets the Font object that should be used as the Font Family's bold-italic member.

```
Boolean
KlgPfFontFamilySetBoldItalic(
   KlgPfObject   font_family,
   KlgPfObject   bold_italic
   )
```

Parameters

font_family: The Font Family object.

bold_italic: Font object to be used as the bold-italic font.

KlgPfFontFamilySetItalic()

Sets the Font object that should be used as the Font Family's italic member.

```
Boolean
KlgPfFontFamilySetItalic(
   KlgPfObject   font_family,
   KlgPfObject   italic
   )
```

Parameters

font_family: The Font Family object.

italic: Font object to be used as the italic font.

KlgPfFontFamilySetNormal()

Sets the Font object that should be used as the Font Family's normal member.

```
Boolean
KlgPfFontFamilySetNormal(
   KlgPfObject   font_family,
   KlgPfObject   normal
   )
```

Parameters

font_family: The Font Family object.

normal: Font object to be used as the normal font.

KlgPfObjectIsFontFamily()

Verifies the object is a Font Family object.

```
Boolean
KlgPfObjectIsFontFamily(
   KlgPfObject   object
   )
```

G

Frame Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Frame object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

G.1 Object Synopsis

Include File: <KlgPfAll.h>, <KlgPfDraw.h> or <KlgPfFrame.h>

Parent Object: Page Object

Usage: `FrameList`, `FlowFrameList`

Description: All text and images are ultimately rendered in a frame. A frame is a child of a Page object. A Frame object can be part of a Document object's main flow or it can render text independent of the main flow. The Frame object provides flow methods similar to the Document object to allow such rendering.

G.2 Attributes (Name, Type, Default)

BorderColor **KlgPfObject** **"Black"**

Specifies the color of the frame's border.

BorderType **KlgPfBorderType KLGPF_BORDER_NONE**

Specifies the type of border the frame should have. Valid types are:

`KLGPF_BORDER_NONE`	No border

KLGPF_BORDER_PLAIN	A single frame border

BorderWidth **double** **0.0**

Specifies the border's thickness in document units. The border is rendered inward. For example, if you set a frame with a 0.25" border, the border will occupy 0.5" of the frame's internal width and 0.5" of the frame's internal height. By default, no text is rendered on the border. However, text can be rendered on the border if the corresponding margin width is set to a value less than the border width.

Color **KlgPfObject** **NULL**

Specifies the background color of the frame.

ColumnCount **int** **1**

Specifies the number of columns in the frame.

ColumnSpacing **double** **0.25**

Specifies the amount of white space between columns.

ColumnWidth **double** ***Dynamic***

Specifies the width of each column. Columns must be of equal widths and are set automatically by PageFormatter. The column width is calculated based on the frame width, `ColumnCount`, `BorderWidth`, the frame's internal `MarginLeft/MarginRight`, and `ColumnSpacing`. Note: if `BorderType` is `KLGPF_BORDER_NONE`, the column width calculation will ignore the `BorderWidth` setting if it is a value other than zero.

LayoutStatus **KlgPfLayoutStatus KLGPF_LAYOUT_COMPLETE**

Indicates the relative readiness of the frame to be printed. Valid values are:

KLGPF_LAYOUT_LEVEL1	The frame has yet to be laid out.
KLGPF_LAYOUT_LEVEL2	The frame's layout is complete except for macros that cannot be resolved until a later time.
KLGPF_LAYOUT_COMPLETE	The frame's layout has been completed and is ready to be printed.

Location **KlgPfRectangle {0.0, 0.0, 0.0, 0.0, KLGPF_UNIT_INCHES}**

A KlgPfRectangle structure that specifies the frame's x and y coordinates (relative to the Page object), width and height values, and what units should be used to interpret the width and height values.

MarginBottom **double** **0.0**

Specifies the amount of internal white space to leave at the bottom of a frame, measured from the frame edge not the frame border. This will most commonly be

used if you're using a border or a colored background and want to establish some "padding" between the text and the frame edge. By default, anytime you set `BorderWidth` PageFormatter will automatically reset this attribute equal to `BorderWidth`. However, if you set `MarginBottom` with *KlgPfFrameSetMarginBottom()*, PageFormatter will cease resetting this attribute.

MarginLeft **double** **0.0**

Specifies the amount of internal white space to leave at the left side of a frame, measured from the frame edge not the frame border. By default, anytime you set `BorderWidth` PageFormatter will automatically reset this attribute equal to `BorderWidth`. However, if you set `MarginLeft` with *KlgPfFrameSetMarginLeft()*, PageFormatter will cease resetting this attribute.

MarginRight **double** **0.0**

Specifies the amount of internal white space to leave at the right side of a frame, measured from the frame edge not the frame border. By default, anytime you set `BorderWidth` PageFormatter will automatically reset this attribute equal to `BorderWidth`. However, if you set `MarginRight` with *KlgPfFrameSetMarginRight()*, PageFormatter will cease resetting this attribute.

MarginTop **double** **0.0**

Specifies the amount of internal white space to leave at the top of a frame, measured from the frame edge not the frame border. By default, anytime you set `BorderWidth` PageFormatter will automatically reset this attribute equal to `BorderWidth`. However, if you set `MarginTop` with *KlgPfFrameSetMarginTop()*, PageFormatter will cease resetting this attribute.

Name **String** ***Dynamic***

Specifies the frame's name (for example "body").

G.3 Methods

KlgPfDrawCircle()

Draws a circle. The line width and line color are defined by the Document's *CurrentDrawStyle* attribute. Returns `True` if successful.

```
Boolean
KlgPfDrawCircle(
   KlgPfObject    frame,
   KlgPfPoint    *center,
   double         radius
   )
```

Parameters

frame: A Frame object.

center: A handle to a `KlgPfPoint` structure. Its x and y members specify the location of the circle's center relative to the frame's top-left corner. The structure's x and y members are in document units.

```
typedef struct {
   double            x, y;
} KlgPfPoint;
```

radius: The circle's radius, in document units.

KlgPfDrawFilledCircle()

Draws a filled circle. The fill color is defined by the Document's `CurrentDrawStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfDrawFilledCircle(
   KlgPfObject    frame,
   KlgPfPoint    *center,
   double         radius
   )
```

Parameters

frame: A Frame object.

center: A handle to a `KlgPfPoint` structure. Its x and y members specify the location of the circle's center relative to the frame's top-left corner. The structure's x and y members are in document units.

```
typedef struct {
   double            x, y;
} KlgPfPoint;
```

radius: The circle's radius, in document units.

KlgPfDrawFilledPolyline()

Uses the given polyline as the perimeter of a filled polygon. If you do not close the polyline, *KlgPfDrawFilledPolyline()* will join the beginning and ending points. The fill color is defined by the Document's `CurrentDrawStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfDrawFilledPolyline(
   KlgPfObject    frame,
   KlgPfPoint    *point_list,
   int            npoints
   )
```

Parameters

frame: A Frame object.

point_list: An array of `KlgPfPoint` structures. *KlgPfDrawFilledPolyline()* draws a line between each point's (x,y) coordinate. Each point is relative to the frame's top-left corner. The coordinates are in document units.

```
typedef struct {
   double           x, y;
} KlgPfPoint;
```

npoints: The number of points.

KlgPfDrawFilledRectangle()

Draws a filled rectangle. The fill color is defined by the Document's `CurrentDrawStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfDrawFilledRectangle(
   KlgPfObject       frame,
   KlgPfRectangle    *location
   )
```

Parameters

frame: A Frame object.

location: A `KlgPfRectangle` structure that specifies the rectangle's x and y coordinates (relative to the frame's top-left corner) and width and height values. Values are in document units.

```
typedef struct {
   double        x, y, width, height;
   KlgPfUnit     unit;
} KlgPfRectangle;
```

KlgPfDrawFilledRoundedRectangle()

Draws a filled rectangle with rounded corners. The fill color is defined by the Document's `CurrentDrawStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfDrawFilledRoundedRectangle(
   KlgPfObject     frame,
   KlgPfRectangle *location,
   double          radius
   )
```

Parameters

frame: A Frame object.

location: A `KlgPfRectangle` structure that specifies the frame's x and y coordinates (relative to the frame's top-left corner) and width and height values. Values are in document units.

```
typedef struct {
   double        x, y, width, height;
   KlgPfUnit     unit;
} KlgPfRectangle;
```

radius: The radius of each corner arc, in document units.

KlgPfDrawPolyline()

Draws multiple line segments. *KlgPfDrawPolyline()* does not automatically join the beginning and ending points. The line width and line color are defined by the Document's `CurrentDrawStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfDrawPolyline(
   KlgPfObject   frame,
   KlgPfPoint    *points,
   int           npoints
   )
```

Parameters

frame: A Frame object.

point_list: An array of `KlgPfPoint` structures. *KlgPfDrawFilledPolyline()* draws a line between each point's (x,y) coordinate. Each point is relative to the frame's top-left corner. The coordinates are in document units.

```
typedef struct {
   double            x, y;
} KlgPfPoint;
```

npoints: The number of points.

KlgPfDrawRectangle()

Draws a rectangle. The line width and line color are defined by the Document's `CurrentDrawStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfDrawRectangle(
   KlgPfObject        frame,
   KlgPfRectangle    *location
   )
```

Parameters

frame: A Frame object.

location: A `KlgPfRectangle` structure that specifies the frame's X and Y coordinates (relative to the frame's top-left corner) and width and height values. Values are in document units.

```
typedef struct {
   double        x, y, width, height;
   KlgPfUnit     unit;
} KlgPfRectangle;
```

KlgPfDrawRoundedRectangle()

Draws a rectangle with rounded corners. The line width and line color are defined by the Document's `CurrentDrawStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfDrawRoundedRectangle(
   KlgPfObject        frame,
   KlgPfRectangle    *location,
   double             radius
   )
```

Parameters

frame: A Frame object.

location: A `KlgPfRectangle` structure that specifies the frame's x and y coordinates (relative to the frame's top-left corner) and width and height values. Values are in document units.

```
typedef struct {
   double        x, y, width, height;
   KlgPfUnit     unit;
} KlgPfRectangle;
```

radius: The radius of each corner arc, in document units. *radius* cannot be greater than half of *height* and *width*.

KlgPfDrawSimpleLine()

Draws a line. The line width and line color are defined by the Document's `CurrentDrawStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfDrawSimpleLine(
   KlgPfObject   frame,
   KlgPfPoint    point1,
   KlgPfPoint    point2
   )
```

Parameters

frame: A Frame object.

point1, point2: `KlgPfPoint` structures which specify the location of the line's beginning and ending. The coordinates are in document units.

```
typedef struct {
   double            x, y;
} KlgPfPoint;
```

KlgPfFrameColumnFeed()

Generates a column break and moves the current text render position to the next column in the frame. If there are no more columns in the frame, *KlgPfFrameColumnFeed()* transfers the flow to the first column in the next frame.

```
Boolean
KlgPfFrameColumnFeed(
   KlgPfObject   frame,
   KlgPfObject   style
   )
```

Returns

Returns `True` if the column feed is successful or `False` on error.

Parameters

frame: A Frame object.

style: A Style object.

KlgPfFrameCopy()

Creates a new frame (using *KlgPfFrameCreate*) with all the same attributes as the *source_frame*. Note that the name of the copy must be different from the name of the *source_frame*.

```
KlgPfObject
KlgPfFrameCopy(
   KlgPfObject   source_frame,
   KlgPfObject   parent,
   String        name
   )
```

Parameters

source_frame: The Frame object you wish to copy.

parent: The new Frame object's parent Page object.

name: Name of the new Frame object (for example, "Body2"). Names must be unique within a page.

KlgPfFrameCreate()

Creates and returns a Frame object and automatically adds itself to its parent's `FrameList`.

```
KlgPfObject
KlgPfFrameCreate(
   KlgPfObject   parent,
   String        name
   )
```

Parameters

parent: A Page object.

name: ASCII name of the Frame object.

KlgPfFrameDestroy()

Destroys a Frame object. Note: destroying the parent object automatically destroys this object. You should not call this method if you have destroyed the parent object.

```
void
KlgPfFrameDestroy(
   KlgPfObject   frame
   )
```

Parameters

frame: The Frame object you wish to destroy.

KlgPfFrameImageFloat()

Draws the specified image on its own line in the specified frame. If PageFormatter calculates there is no room left in the frame to render the image in full, it will not render the image. This method is meant for "non-flow" frames, such as headers and footers.

```
Boolean
KlgPfFrameImageFloat(
   KlgPfObject frame,
   KlgPfObject style,
   KlgPfObject image
   )
```

Parameters

frame: The frame to which you wish to render the image

style: A Style object you wish to apply to the rendered image.

image: An Image object.

KlgPfFrameImageEmbed()

Draws the specified image as an inline graphic in the specified frame. If PageFormatter calculates there is no space left on the line to render the image in full, it will wrap the line and render the image on the next line. If there is no next line or if the image does not fit on the next line, it is not rendered. This method is meant for "non-flow" frames, such as headers and footers.

```
Boolean
KlgPfFrameImageEmbed(
   KlgPfObject frame,
   KlgPfObject style,
   KlgPfObject image,
   KlgPfImageAlignment alignment
   )
```

Parameters

frame: The frame to which you wish to render the image

style: A Style object which will aid in rendering the image.

image: An Image object.

alignment: Images may be laid out, relative to the text in the line, using one of following alignments:

KLGPF_IMAGE_BASELINE	Aligns the image at the text's baseline.
KLGPF_IMAGE_BOTTOM	Aligns the image at the bottom of the line.
KLGPF_IMAGE_CENTER	Vertically centers the image in the line.
KLGPF_IMAGE_TOP	Aligns the image at the top of the line.

KlgPfFrameImagePlaceXY()

Renders the specified image at the passed X-Y coordinates. This has no effect or regard for the current flow.

```
Boolean
KlgPfFrameImagePlaceXY(
   KlgPfObject frame,
   KlgPfObject style,
   KlgPfObject image,
   KlgPfPoint *location
)
```

Parameters

frame: The Frame object to which to render.

style: The Style object you wish to apply to the rendered string.

image: An Image object.

location: A KlgPfPoint structure specifies the origin (upper-left corner) of the Image in the given frame.

```
typedef struct {
   double        x,y;
} KlgPfPoint;
```

KlgPfFrameGetBorderColor()

Returns the frame's border Color object.

```
KlgPfObject
KlgPfFrameGetBorderColor(
   KlgPfObject   frame
   )
```

KlgPfFrameGetBorderType()

Returns the Frame object's border type.

```
KlgPfBorderType
KlgPfFrameGetBorderType(
   KlgPfObject   frame
   )
```

KlgPfFrameGetBorderWidth()

Returns the frame's border width.

```
double
KlgPfFrameGetBorderWidth(
   KlgPfObject   frame
   )
```

KlgPfFrameGetColor()

Returns the Frame object's background Color object.

```
KlgPfObject
KlgPfFrameGetColor(
   KlgPfObject   frame
   )
```

KlgPfFrameGetColumnCount()

Returns the number of columns in the frame.

```
int
KlgPfFrameGetColumnCount(
   KlgPfObject   frame
   )
```

KlgPfFrameGetColumnSpacing()

Returns the amount of white space in document units between columns.

```
double
KlgPfFrameGetColumnSpacing(
   KlgPfObject   frame
   )
```

KlgPfFrameGetColumnWidth()

Returns the width of each column. Columns are always equal widths.

```
double
KlgPfFrameGetColumnWidth(
   KlgPfObject   frame
   )
```

KlgPfFrameGetCurrentAscender()

Retrieves the ascender (that is, the space above the baseline) of the current line being rendered (this includes images). The value returned is in document units.

```
double
KlgPfFrameGetCurrentAscender(
   KlgPfObject   frame
   )
```

KlgPfFrameGetCurrentBaseline()

Retrieves the baseline of the current line being rendered (this includes images). The value returned is in document units.

```
double
KlgPfFrameGetCurrentBaseline(
   KlgPfObject   frame
   )
```

KlgPfFrameGetCurrentDescender()

Retrieves the descender (that is, the space below the baseline) of the current line being rendered (this includes images). The value returned is in document units.

```
double
KlgPfFrameGetCurrentDescender(
   KlgPfObject   frame
   )
```

KlgPfFrameGetCurrentX()

Returns the X position, relative to the frame or columns left edge, to which PageFormatter is currently rendering.

```
double
KlgPfFrameGetCurrentX(
   KlgPfObject   frame
   )
```

KlgPfFrameGetCurrentY()

Returns the Y position, relative to the frame or columns top edge, to which PageFormatter is currently rendering.

```
double
KlgPfFrameGetCurrentY(
   KlgPfObject   frame
   )
```

KlgPfFrameGetHeight()

Returns the Frame object's height.

```
double
KlgPfFrameGetHeight(
   KlgPfObject   frame
   )
```

KlgPfFrameGetLayoutStatus()

Returns the frame's relative readiness to be printed.

```
KlgPfLayoutStatus
KlgPfFrameGetLayoutStatus(
   KlgPfObject   frame
   )
```

KlgPfFrameGetMarginBottom()

Returns the amount of internal white space at the bottom of the frame.

```
double
KlgPfFrameGetMarginBottom(
   KlgPfObject   frame
   )
```

KlgPfFrameGetMarginLeft()

Returns the amount of internal white space at the left side of the frame.

```
double
KlgPfFrameGetMarginLeft(
   KlgPfObject   frame
   )
```

KlgPfFrameGetMarginRight()

Returns the amount of internal white space at the right side of the frame.

```
double
KlgPfFrameGetMarginRight(
   KlgPfObject   frame
   )
```

KlgPfFrameGetMarginTop()

Returns the amount of internal white space at the top of the frame.

```
double
KlgPfFrameGetMarginTop(
   KlgPfObject   frame
   )
```

KlgPfFrameGetName()

Returns the name of the Frame object (for example "body").

```
String
KlgPfFrameGetName(
   KlgPfObject   frame
   )
```

KlgPfFrameGetNextTab()

Returns the next Tab object that would be used by the specified Style object if *KlgPfFlowTab()* were called.

```
KlgPfObject
KlgPfFrameGetNextTab(
   KlgPfObject   frame,
   KlgPfObject   style
   )
```

KlgPfFrameGetParent()

Returns the Frame object's parent Page object.

```
KlgPfObject
KlgPfFrameGetParent(
   KlgPfObject   frame
   )
```

KlgPfFrameGetWidth()

Returns the Frame object's width.

```
double
KlgPfFrameGetWidth(
   KlgPfObject   frame
   )
```

KlgPfFrameGetX()

Returns the Frame object's X position relative to the Page object.

```
double
KlgPfFrameGetX(
   KlgPfObject   frame
   )
```

KlgPfFrameGetY()

Returns the Frame object's Y position relative to the Page object.

```
double
KlgPfFrameGetY(
   KlgPfObject   frame
   )
```

KlgPfFrameHRuleLength()

Draws a horizontal ruling line in the column, of a specified length.

```
Boolean
KlgPfFrameHRuleLength(
   KlgPfObject   frame,
   KlgPfObject   style,
   KlgPfObject   draw_style,
   double        length
)
```

Parameters

frame: A Frame object.

style: A Style object you wish to apply to the rendered image.

draw_style: A DrawStyle object, defining the line's height and color.

length: The rule's length, in document units. If wider than the column, the rule will be truncated to the column's width.

KlgPfFrameHRulePercent()

Draws a horizontal ruling line in the column. The line's length is expressed as a percentage of the column width.

```
Boolean
KlgPfFrameHRulePercent(
   KlgPfObject   frame,
   KlgPfObject   style,
   KlgPfObject   draw_style,
   double        percent
)
```

Parameters

frame: A Frame object.

style: A Style object you wish to apply to the rendered image.

draw_style: A DrawStyle object, defining the line's height and color.

percentage: The rule's length, expressed as a percentage of the column width.

KlgPfFrameLineFeed()

Generates a line feed and resets the render position to the start of the next line in the frame. It also causes the start of a new paragraph within the frame.

```
Boolean
KlgPfFrameLineFeed(
   KlgPfObject   frame,
   KlgPfObject   style
   )
```

KlgPfFramePrint()

Renders the specified string. *KlgPfFramePrint()* differs from *KlgPfFlowPrint()* in that *KlgPfFramePrint()* renders to a frame specified in the method's parameter list. *KlgPfFlowPrint()* renders to the frame identified by the Document object's `CurrentFrame` attribute. *KlgPfFramePrint()* is useful for printing text that is not part of the main flow (for example, header/footer text). If there is no room for some or all of the text in the frame, the text will not be rendered.

```
Boolean
KlgPfFramePrint(
   KlgPfObject   frame,
   KlgPfObject   style,
   String        text
   )
```

Returns

Returns `True` if the flow is rendered successfully.

Parameters

frame: The Frame object to which to render.

style: The Style object you wish to apply to the rendered string.

text: A string of text to be rendered.

KlgPfFramePrintXY()

Renders the specified text at the passed X-Y coordinates. This has no effect or regard for the current flow.

```
Boolean
KlgPfFramePrintXY(
   KlgPfObject   frame,
   KlgPfObject   style,
   String        str,
   double        x,
   double        y
   )
```

KlgPfFrameSetBorderColor()

Sets the frame's `BorderColor` attribute.

```
Boolean
KlgPfFrameSetBorderColor(
   KlgPfObject   frame,
   KlgPfObject   color
   )
```

Returns

Returns `True` if the attribute is set successfully.

Parameters

frame: A Frame object.

color: A Color object.

KlgPfFrameSetBorderType()

Sets the frame's `BorderType` attribute.

```
Boolean
KlgPfFrameSetBorderType(
   KlgPfObject       frame,
   KlgPfBorderType   border_type
   )
```

Parameters

frame: A Frame object.

border_type: The type of border the frame should have (for example, `KLGPF_BORDER_PLAIN`).

KlgPfFrameSetBorderWidth()

Sets the Frame border's thickness.

```
Boolean
KlgPfFrameSetBorderWidth(
   KlgPfObject   frame,
   double        width
   )
```

KlgPfFrameSetColor()

Sets the Frame object's background Color object.

```
Boolean
KlgPfFrameSetColor(
   KlgPfObject   frame,
   KlgPfObject   color
   )
```

KlgPfFrameSetColumnCount()

Sets the number of columns in the frame. Setting the column count forces a recalculation of the `ColumnWidth` attribute. Note: you cannot set a `ColumnCount` that results in a computed `ColumnWidth` less than or equal to 0.

```
Boolean
KlgPfFrameSetColumnCount(
   KlgPfObject   frame,
   int           count
   )
```

KlgPfFrameSetColumnSpacing()

Sets the amount of white space between columns. Setting the column spacing forces a recalculation of the `ColumnWidth` attribute. Note: you cannot set column spacing greater than the width of the frame.

```
Boolean
KlgPfFrameSetColumnSpacing(
   KlgPfObject   frame,
   double        spacing
   )
```

KlgPfFrameSetCurrentX()

Sets the exact X position, relative to the current column's left edge, to which PageFormatter should render text.

```
Boolean
KlgPfFrameSetCurrentX(
   KlgPfObject   frame,
   double        x
   )
```

KlgPfFrameSetCurrentY()

Sets the exact Y position, relative to the current column's top edge, to which PageFormatter should render text.

```
Boolean
KlgPfFrameSetCurrentY(
   KlgPfObject   frame,
   double        y
   )
```

KlgPfFrameSetHeight()

Sets the Frame object's height.

```
Boolean
KlgPfFrameSetHeight(
   KlgPfObject   frame,
   double        height
   )
```

KlgPfFrameSetLayoutStatus()

Sets frame's `LayoutStatus` attribute (for example, `KLGPF_LAYOUT_COMPLETE`). This method should be used with care, since it may affect macro evaluation.

```
Boolean
KlgPfFrameSetLayoutStatus(
   KlgPfObject       frame,
   KlgPfLayoutStatus status
   )
```

KlgPfFrameSetMarginBottom()

Sets the amount of white space between the text and the bottom of the frame.

```
Boolean
KlgPfFrameSetMarginBottom(
   KlgPfObject   frame,
   double        bottom_margin
   )
```

KlgPfFrameSetMarginLeft()

Sets the amount of white space between the first column and the left side of the frame.

```
Boolean
KlgPfFrameSetMarginLeft(
   KlgPfObject   frame,
   double        left_margin
   )
```

KlgPfFrameSetMarginRight()

Sets the amount of white space between the first column and the right side of the frame.

```
Boolean
KlgPfFrameSetMarginRight(
   KlgPfObject   frame,
   double        right_margin
   )
```

KlgPfFrameSetMarginTop()

Sets the amount of white space between the first column and the top of the frame.

```
Boolean
KlgPfFrameSetMarginTop(
   KlgPfObject   frame,
   double        top_margin
   )
```

KlgPfFrameSetWidth()

Sets the Frame object's width.

```
Boolean
KlgPfFrameSetWidth(
   KlgPfObject   frame,
   double        width
   )
```

KlgPfFrameSetX()

Sets the Frame object's x offset relative to its parent Page object's upper left corner.

```
Boolean
KlgPfFrameSetX(
   KlgPfObject   frame,
   double        x
   )
```

KlgPfFrameSetY()

Sets the Frame object's y offset relative to its parent Page object's upper left corner.

```
Boolean
KlgPfFrameSetY(
   KlgPfObject   frame,
   double        y
   )
```

KlgPfFrameTab()

Generates a tab feed in the specified frame flow. *KlgPfFrameTab()* differs from *KlgPfFlowTab()* in that *KlgPfFrameTab()* generates the tab feed in a frame specified by the method's parameter list. *KlgPfFlowTab()* generates the tab feed in the frame identified by the Document object's `CurrentFrame` attribute. *KlgPfFrameTab()* is useful for generating tabs in text that is not part of the main flow (for example, header/footer text). You must pass it a Frame and Style object.

```
Boolean
KlgPfFrameTab(
   KlgPfObject   frame,
   KlgPfObject   style
   )
```

KlgPfObjectIsFrame()

Returns `True` if the object is a Frame object, otherwise `False`.

```
Boolean
KlgPfObjectIsFrame(
   KlgPfObject   object
   )
```

H

Image Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Image object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

H.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfImage.h>
Include <KlgPfGIF.h> for GIF images and <KlgPfJPEG.h> JPEG images.

Description: The Image object provides attributes and methods for creating an image. **Note:** You must destroy all Image objects you create.

H.2 Attributes (Name, Type, Default)

Size **KlgPfRectangle**

The `KlgPfRectangle` structure's height and width members specify the printed size of the image. The image will be scaled to these dimensions on output except in the X11 driver, where the image will be centred in a rectangular area of this size. This attribute is required before an image can be printed.

```
typedef struct {
   double           x, y, width, height;
   KlgPfUnit        unit;
} KlgPfRectangle;
```

Type **KlgPfImageType** **KLGPF_IMAGE_TYPE_XPM**
Specifies the image object's format. Valid values are:

KLGPF_IMAGE_TYPE_XPM	XPM image
KLGPF_IMAGE_TYPE_EPS	EPS image
KLGPF_IMAGE_TYPE_JPEG	JPEG image
KLGPF_IMAGE_TYPE_GIF	GIF image

H.3 Methods

KlgPfImageCopy()

Returns a copy of Image object.

```
KlgPfObject
KlgPfImageCopy(
   KlgPfObject       source_image
   )
```

KlgPfImageCreateFromEPSF()

Returns an EPS image object.

```
KlgPfObject
KlgPfImageCreateFromEPSF(
   FILE              *input_file
   )
```

Parameters

input_file: A handle to an open input file stream for the EPS image.

KlgPfImageCreateFromGIF()

Returns a GIF image object. You must include `<KlgPfGIF.h>`.

```
KlgPfObject
KlgPfImageCreateFromGIF(
   FILE              *input_file
   )
```

Parameters

image: A handle to an open input file stream for the GIF image.

KlgPfImageCreateFromJPEG()

Returns a JPEG image object. You must include `<KlgPfJPEG.h>`.

```
KlgPfObject
KlgPfImageCreateFromJPEG(
   FILE              *input_file
   )
```

Parameters

image: A handle to an open input file stream for the JPEG image.

KlgPfImageCreateFromXPM()

Returns an XPM image object.

```
KlgPfObject
KlgPfImageCreateFromXPM(
   XpmImage          image
   )
```

Parameters

image: A handle to an XpmImage structure for an XPM image.

KlgPfImageDestroy()

Destroys an image object. You must destroy all image objects you create.

```
void
KlgPfImageDestroy(
   KlgPfObject       image_object
   )
```

KlgPfImageGetSize()

Retrieves the Image object's `Size` attribute. *KlgPfImageGetSize()* returns a `KlgPfRectangle` structure.

```
KlgPfRectangle *
KlgPfImageGetSize(
   KlgPfObject       image_object
   )
```

KlgPfImageGetVisibleBoundingBox()

Returns a Boolean value indicating if the bounding box within which the image is located is to be filled in grey. This function is useful for images displayed with the X11 previewer, where the image can't be scaled to its specified size, and for EPS images which can't be displayed in X11, Acrobat, or PCL.

```
Boolean
KlgPfImageGetVisibleBoundingBox(
   KlgPfObject       image
   )
```

KlgPfImageSetSize()

Sets the `Size` attribute. This attribute must be set for an image to be flowed into the document.

```
Boolean
   KlgPfImageSetSize(
   KlgPfObject        image_object,
   KlgPfRectangle    *dimensions
   )
```

Parameters

image_object: The Image object.

dimensions: A KlgPfRectangle structure whose height and width members specify the size at which to display the image. The image will be scaled (where possible) to this

size when being printed. The unit member specifies the units of measurement (for example, if KLGPF_UNIT_INCHES is set, then the width and height members are interpreted as inches). It is not necessary for the units of the image to match the units specified for the document.

```
typedef struct {
   double            x, y, width, height;
   KlgPfUnit         unit;
} KlgPfRectangle;
```

KlgPfImageSetVisibleBoundingBox()

Set the visible bounding box flag to specify whether the bounding box within which the image is located is to be filled in grey. This function returns True on success.

```
Boolean
   KlgPfImageGetVisibleBoundingBox(
   KlgPfObject       image,
   Boolean           is_visible
   )
```

KlgPfObjectIsImage()

Returns `True` if the object is an Image object.

```
Boolean
KlgPfObjectIsImage(
   KlgPfObject       object
   )
```

I

List Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the List object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

I.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfList.h>

Parent Object: Document Object, Page Object, Printer Object, Style Object

Usage: Document object: `PageList`, `PageTemplateList`, `StyleList`
Page object: `FrameList`, `FlowFrameList`
Printer object: `FontFamilyList`, `FontList`, `PrinterOptionList`, `PrinterOptionSelectedList`, `ResolutionSupportedList`
Style object: `TabList`

Description: The List object provides a convenient way of storing groups of items. Items can be added, deleted, sorted, and retrieved using methods defined for these purposes. Note: the list specified by an List object is zero-based: the first element has index 0, the second element has index 1, and so on.

I.2 Attributes (Name, Type, Default)

ChangedProc **KlgPfListItemChangedProc** ***Dynamic***

Specifies the change procedure to be used by a callback when a list item is added, deleted, or destroyed. The attribute is of type `KlgPfListChangedProc` and has the following prototype:

```
typedef void (*KlgPfListChangedProc) (
    KlgPfObject                    list,
    KlgPfListChangedCallbackStruct  *call_data
    );
```

CompareProc **KlgPfListItemCompareProc** ***Dynamic***

Specifies the comparison procedure to be used in sorting and finding items on the list. The attribute is of type `KlgPfListItemCompareProc` and has the following prototype:

```
typedef int (*KlgPfListItemCompareProc) (
        XtPointer  item1,
        XtPointer  item2
        );
```

DestroyProc **KlgPfListItemDestroyProc** ***Dynamic***

Specifies the procedure that the list object uses to destroy its items (if they were allocated in some special fashion). The item-destroying procedure must be defined to be of the following type:

```
typedef void (*KlgPfListItemDestroyProc) (
   KlgPfObject  object,
   XtPointer    item,
   XtPointer    user_data
   );
```

ItemCount **int** **0**

Specifies the number of items in the list.

ItemSize **size_t** ***Dynamic***

Specifies the size of the items in the list.

Items **XtPointer** ***Dynamic***

A pointer to the array of items in the List object.

Sorted **Boolean** **False**

Specifies whether the items in the list should be sorted or not.

UserData **XtPointer** **NULL**

A handle for developer-defined user data associated with the list.

I.3 Methods

KlgPfListAdd()

Adds an item to the list in its proper, sorted position. The sort order is defined by the procedure returned by *KlgPfListGetCompareProc()*.

```
int
KlgPfListAdd(
   KlgPfObject   list,
   XtPointer     item
   )
```

Returns

Returns the position index of the new node or `KLGPF_LIST_ITEM_NOT_FOUND` if the list is not sorted.

Parameters

list: A list object.

item: A pointer to the new item.

KlgPfListAppend()

Appends an item to the end of the list.

```
int
KlgPfListAppend(
   KlgPfObject   list,
   XtPointer     item
   )
```

Returns

Returns the position index.

Parameters

list: A list object.

item: A pointer to the new item.

KlgPfListCreate()

Initializes a list of items of the given size.

```
KlgPfObject
KlgPfListCreate(
   size_t        size
   )
```

Returns

Returns a List object or `NULL` on error.

Parameters

size: The intended bit size of the items in the list.

KlgPfListDelete()

Deletes the item at the specified position.

```
int
KlgPfListDelete(
   KlgPfObject   list,
   int           position
   )
```

Returns

Returns the position of the deleted item or `KLGPF_LIST_ITEM_NOT_FOUND`.

Parameters

list: A list object.

position: The position of the item to be deleted.

KlgPfListDeleteAll()

Deletes all the items from the list.

```
void
KlgPfListDeleteAll(
   KlgPfObject   list
   )
```

Parameters

list: A list object.

KlgPfListDestroy()

Destroys the specified list. Note: if the list you are destroying was retrieved from its parent object, destroying the parent object automatically destroys this object. You should not call this method if you have destroyed the parent object.

```
void
KlgPfListDestroy(
   KlgPfObject   list
   )
```

Parameters

list: A list object.

KlgPfListDestroyObject()

Destroys an object in the specified list.

```
void
KlgPfListDestroyObject(
   KlgPfObject    list,
   KlgPfObject   *object,
   XtPointer      user_data
   )
```

Parameters

list: A list object.

object: The object to be destroyed.

user_data: This should always be `NULL`.

KlgPfListDestroyXtPointer()

Destroys a non-object item in the specified list.

```
void
KlgPfListDestroyXtPointer(
   KlgPfObject   list,
   XtPointer     *item
   )
```

Parameters

list: A list object.

item: The item to be destroyed.

KlgPfListDuplicate()

Makes a copy of a list. Note: The new list's item destroy procedure is set to `NULL`; it can be set using *KlgPfListSetDestroyProc()*.

```
KlgPfObject
KlgPfListDuplicate(
   KlgPfObject   list
   )
```

Returns

Returns a list object or `NULL` on error.

Parameters

list: A list object.

KlgPfListFind()

Finds a list item matching a specified item (using the procedure returned by *KlgPfListGetCompareProc()* to check for equality).

```
int
KlgPfListFind(
   KlgPfObject   list,
   XtPointer     item
   )
```

Returns

Returns the position index of the item if found or `KLGPF_LIST_ITEM_NOT_FOUND` on error.

Parameters

list: A list object.

item: A pointer to the item to be found.

KlgPfListGetChangedProc()

Retrieves the procedure that the list callback uses when a list item has been added, deleted, or destroyed.

```
KlgPfListChangedProc
KlgPfListGetChangedProc(
   KlgPfObject   list_object
   )
```

Returns

Returns a `KlgPfListChangedProc` structure.

KlgPfListGetCompareProc()

Retrieves the procedure that the list uses to compare elements for searching and sorting.

```
KlgPfListItemCompareProc
KlgPfListGetCompareProc(
   KlgPfObject   list
   )
```

Returns

Returns a `KlgPfListItemCompareProc` structure.

KlgPfListGetItem()

Returns the item at the specified index position or `NULL` on error.

```
XtPointer
KlgPfListGetItem(
   KlgPfObject   list,
   int           position
   )
```

KlgPfListGetItemCount()

Returns the number of items stored in the specified list or `-1` on error.

```
int
KlgPfListGetItemCount(
   KlgPfObject   list
   )
```

KlgPfListGetItems()

Returns an array of items in the list object or `NULL` if the list is empty.

```
XtPointer
KlgPfListGetItems(
   KlgPfObject   list
   )
```

KlgPfListGetItemSize()

Returns the size of an item in the list or `-1` on error.

```
int
KlgPfListGetItemSize(
   KlgPfObject   list
   )
```

KlgPfListGetUserData()

Returns the user data associated with the List object.

```
XtPointer
KlgPfListGetUserData(
   KlgPfObject   list
   )
```

KlgPfListInsertAfter()

Inserts a new item into a list *after* a specified position. The new item must be the same size as the value returned by *KlgPfListGetItemSize()*.

```
int
KlgPfListInsertAfter(
   KlgPfObject   list,
   int           position,
   XtPointer     item
   )
```

Returns

Returns the index of the new node or `KLGPF_LIST_INVALID_POSITION` on failure.

Parameters

list: A list object.

position: The position after which the new item is to be added.

item: A pointer to the new item.

KlgPfListInsertBefore()

Inserts a new item into a list *before* a specified position. If the position is greater than the number of elements in the list, the item is appended to the end of the list. The new item must be the same size as the value returned by *KlgPfListGetItemSize()*.

```
int
KlgPfListInsertBefore(
   KlgPfObject   list,
   int           position,
   XtPointer     item
   )
```

Returns

Returns the index of the new node or `KLGPF_LIST_INVALID_POSITION` on failure.

Parameters

list: A list object.

position: The position after which the new item is to be added.

item: A pointer to the new item.

KlgPfListIsList()

Returns `True` if the object is a List object, otherwise `False`.

```
Boolean
KlgPfListIsList(
   KlgPfObject   list
   )
```

KlgPfListIsSorted()

Returns a non-zero value if the list is sorted.

```
int
KlgPfListIsSorted(
   KlgPfObject   list
   )
```

KlgPfListLookupItem()

Locates the item in the specified list. This routine performs a strict memory compare and does not use the procedure returned by *KlgPfListGetCompareProc()*.

```
int
KlgPfListLookupItem(
   KlgPfObject   list,
   XtPointer     item
   )
```

Returns

Returns the position of the matched item, or `KLGPF_LIST_ITEM_NOT_FOUND` on failure.

Parameters

list: A list object.

item: A pointer to the item.

KlgPfListMoveItemsAfter()

Moves a specified number of items from a source location to *after* the target location.

```
Boolean
KlgPfListMoveItemsAfter(
   KlgPfObject   list,
   int           target_position,
   int           source_position,
   int           number_of_items_to_move
   )
```

Returns

Returns `True` if the move is successful.

Parameters

list: A list object.

target_position: The destination location; the moved items are placed *after* this index location.

source_position: The location of the first item to move.

number_of_items_to_move: The number of items to be moved.

KlgPfListMoveItemsBefore()

Moves a specified number of items from a source location to *before* the target location.

```
Boolean
KlgPfListMoveItemsBefore(
   KlgPfObject   list,
   int           target_position,
   int           source_position,
   int           number_of_items_to_move
   )
```

Returns

Returns `True` if the move is successful.

Parameters

list: A list object.

target_position: The destination location; the moved items are placed *before* this index location.

source_position: The location of the first item to move.

number_of_items_to_move: The number of items to be moved.

KlgPfListRemove()

Removes the item at the specified position from a list. The destroy procedure (registered by calling *KlgPfListSetDestroyProc()*) is not called when the item is removed.

```
int
KlgPfListRemove(
   KlgPfObject   list,
   int           position
   )
```

Returns

Returns the position of the removed item.

Parameters

list: A list object.

position: The position of the item to be removed.

KlgPfListRemoveAll()

Removes all the items from a list. The destroy procedure (registered by calling *KlgPfListSetDestroyProc()*) is not called when the item is removed.

```
void
KlgPfListRemoveAll(
   KlgPfObject   list
   )
```

Parameters

list: A list object.

KlgPfListSearchForward()

Searches *forward* in a list, from a specified location, for a match of a specified item. The procedure returned by *KlgPfListGetCompareProc()* is used to perform comparisons.

```
int
KlgPfListSearchForward(
   KlgPfObject   list,
   int           start_position,
   XtPointer     item
   )
```

Returns

Returns the position of the matched item, or `KLGPF_LIST_ITEM_NOT_FOUND` on failure.

Parameters

list: A list object.

start_position: The position of the first list item to be compared.

item: A pointer to the item to be matched.

KlgPfListSearchReverse()

Searches *backward* in a list, from a specified location, for a match for a specified item. (The procedure returned *by KlgPfListGetCompareProc()* is used to perform comparisons.)

```
int
KlgPfListSearchReverse(
   KlgPfObject   list,
   int           start_position,
   XtPointer     item
   )
```

Returns

Returns the position of the matched item, or `KLGPF_LIST_ITEM_NOT_FOUND` on failure.

Parameters

list: A list object.

start_position: The position of the first list item to be compared.

item: A pointer to the item to be matched.

KlgPfListSetChangedProc()

Specifies the procedure that a list callback object uses when a list item has been added, removed, or deleted.

```
void
KlgPfListChangedProc(
   KlgPfObject              list,
   KlgPfListItemChangedProc changed_proc
   )
```

The change procedure must be defined to be of the following type:

```
typedef void (*KlgPfListChangedProc) (
    KlgPfObject                    list,
    KlgPfListChangedCallbackStruct  *call_data
    );
```

List is the List object. *Call_data* is a handle to a `KlgPfListChangedCallbackStruct`. The structure is defined as:

```
typedef struct {
   KlgPfCallbackReason          reason;
   XtPointer                    user_data;
   XtPointer                    modified_item;
   int                          position; /* position of change */
} KlgPfListChangedCallbackStruct;
```

KlgPfListSetCompareProc()

Specifies the procedure that a list object uses to compare list elements for sorting and searching.

```
void
KlgPfListSetCompareProc(
   KlgPfObject              list,
   KlgPfListItemCompareProc compare_proc
   )
```

The compare procedure must be defined to be of the following type:

```
typedef int (*KlgPfListItemCompareProc) (
        XtPointer  item1,
        XtPointer  item2
        );
```

Item1 and *item2* are the two items to be compared. The compare procedure will return the following:

- a positive integer if item1 is greater than item2
- 0 if the two items are equal
- a negative integer if item1 is less than item2

Parameters

list: A list object.

compare_proc: The procedure to be called.

KlgPfListSetDestroyProc()

Specifies the procedure that the list object uses to destroy its items (if they were not allocated by the list).

```
void
KlgPfListSetDestroyProc(
   KlgPfObject   list,
   KlgPfListItemDestroyProcdestroy_proc
   )
```

The item-destroying procedure must be defined according to the following prototype:

```
typedef void (*KlgPfListItemDestroyProc)(
       KlgPfObject       object,
       XtPointer         item,
       XtPointer         user_data
       );
```

Parameters

list: A list object.

destroy_proc: The procedure to be called.

KlgPfListSetItem()

Sets an item in a list. The old element is destroyed and the new element is copied in its place.

```
Boolean
KlgPfListSetItem(
   KlgPfObject   list,
   int           position,
   XtPointer     item
   )
```

Returns

Returns `False` if the index specified is invalid.

Parameters

list: A list object.

position: The location of the item to be set.

item: A pointer to the item to be set in the list.

KlgPfListSetUserData()

The List object enables you to define user data to be associated with the list. To define user data, pass a pointer to the user data to the *KlgPfListSetUserData()* method. The developer is responsible for freeing the memory used by the user data.

```
void
KlgPfListSetUserData(
   KlgPfObject   list,
   XtPointer     user_data
   )
```

Parameters

list: A List object.

user_data: A pointer to the user data.

KlgPfListSort()

Sorts the items in a list object. If the list is already sorted, this method does nothing.

```
void
KlgPfListSort(
   KlgPfObject   list
   )
```

J

Page Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Page object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

J.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfPage.h>

Parent Object: Document Object

Usage: `PageList`, `PageTemplateList`

Description: The Page object is a child of the Document object. The Page object's attributes specify the layout of a page, what Frame objects exist on the page, and how the flow moves through the Frame objects.

J.2 Attributes (Name, Type, Default)

Color **KlgPfObject** **"White"**

Specifies the background color of the page.

FlowFrameList **KlgPfObject** ***Dynamic***

The order the flow moves through the frames on the page. When the text flow reaches the bottom of a frame, the flow is transferred to the next Frame object on the `FlowFrameList`. When the `FlowFrameList` is exhausted the flow is transferred to the first frame of the `FlowFrameList` on the next page.

FlowPageTemplate **String** **KLGPF_TEMPLATE_SAME**

The page template to use when *KlgPfFlowFormFeed()* is called or a page flow event occurs. A new page is then created from this template and flow is transferred to this page. Be default this attribute indicates the template that was used to create the current page.

FlowSectionTemplate **String** **KLGPF_TEMPLATE_NEXT_IN_SEQUENCE**

The page template to use when *KlgPfFlowSectionBreak()* is called. A new page is then created from this template and the flow is transferred to this page. By default the template that is next in sequence on the documents `PageTemplateList` is used.

FrameList **KlgPfObject** ***Dynamic***

A list of all Frame objects on the page, including frames not part of the regular flow as defined by the `FlowFrameList` attribute (for example header and footer frames).

LayoutStatus **KlgPfLayoutStatus** **KLGPF_LAYOUT_LEVEL1**

Indicates the relative readiness of the page to be printed. Valid values are:

`KLGPF_LAYOUT_LEVEL1`	The page is not laid out.
`KLGPF_LAYOUT_LEVEL2`	The page's layout is complete except for macros that cannot be resolved until a later time. For example, a macro that includes the total number of pages in the page layout cause the page to be left at a `KLGPF_LAYOUT_LEVEL2` until it is resolved.
`KLGPF_LAYOUT_COMPLETE`	The page's layout has been completed and is ready to be printed.

Location **KlgPfRectangle** **{0.0, 0.0, 0.0, 0.0, KLGPF_UNIT_INCHES}**

A KlgPfRectangle structure that specifies the page's beginning X and Y coordinates (relative to the Document object), and width and height values.

Name **String** ***Dynamic***

Name of the Page object or template Page object.

Type **KlgPfPageType** ***Dynamic***

A flag to differentiate a printable Page object from a template Page object. Valid values are:

`KLGPF_PAGE_ACTUAL`	Indicates the Page object is an actual rendered page.
`KLGPF_PAGE_REFERENCE`	Indicates the Page object is an actual rendered page copied to disk to free memory.

KLGPF_PAGE_TEMPLATE	Indicates the Page object is a page template.
KLGPF_PAGE_DELETED	Indicates the Page object was deleted when using a FlushPolicy of KLGPF_FLUSH_ON_OUTPUT.

J.3 Methods

KlgPfObjectIsPage()

Returns True if the object is a Page object, otherwise False.

```
Boolean
KlgPfObjectIsPage(
   KlgPfObject      object
   )
```

KlgPfPageCopy()

Returns a Page object which is a copy of the given page.

```
KlgPfObject
KlgPfPageCopy(
   KlgPfObject      source,
   String           copy_name,
   KlgPfPageType    type
   )
```

Parameters

source: The Page object you wish to copy.

copy_name: The name attribute of the copied Page object.

type: Type attribute of the new Page object (KLGPF_PAGE_ACTUAL if it's a printable Page object or KLGPF_PAGE_TEMPLATE if it's a template Page object).

KlgPfPageCreate()

Creates a Page object.

```
KlgPfObject
KlgPfPageCreate(
   KlgPfObject      parent,
   String           name,
   KlgPfPageType    type
   )
```

Parameters

parent: The Page object's parent Document object.

name: A string name for the page (for example, "mypage").

type: Type attribute of the new Page object (KLGPF_PAGE_ACTUAL if it's a printable Page object or KLGPF_PAGE_TEMPLATE if it's a template Page object).

KlgPfPageDestroy()

Destroys a Page object. Note: destroying the parent object automatically destroys this object. You should not call this method if you have destroyed the parent object.

```
void
KlgPfPageDestroy(
   KlgPfObject      page
   )
```

KlgPfPageGetColor()

Returns the Page object's background Color object.

```
KlgPfObject
KlgPfPageGetColor(
   KlgPfObject      page
   )
```

KlgPfPageGetFlowFrameList()

Returns the list of the order the flow moves through the frames on the page.

```
KlgPfObject
KlgPfPageGetFlowFrameList(
   KlgPfObject      page
   )
```

KlgPfPageGetFlowPageTemplate()

Returns the name of the template to use to create the next page when *KlgPfFlowFormFeed()* is called or a page flow event occurs.

```
String
KlgPfPageGetFlowPageTemplate(
   KlgPfObject      page
   )
```

KlgPfPageGetFlowSectionTemplate()

Returns the name of the template to use to create the next page when *KlgPfFlowSectionBreak()* is called.

```
String
KlgPfPageGetFlowSectionTemplate(
   KlgPfObject      page
   )
```

KlgPfPageGetFrameList()

Returns the list of all frames on the page.

```
KlgPfObject
KlgPfPageGetFrameList(
   KlgPfObject      page
   )
```

KlgPfPageGetHeight()

Returns the Page object's height measured in document units.

```
double
KlgPfPageGetHeight(
   KlgPfObject      page
   )
```

KlgPfPageGetLayoutStatus()

Returns the Page object's layout status.

```
KlgPfLayoutStatus
KlgPfPageGetLayoutStatus(
   KlgPfObject      page
   )
```

KlgPfPageGetName()

Returns the name of the Page object.

```
String
KlgPfPageGetName(
   KlgPfObject      page
   )
```

KlgPfPageGetParent()

Returns the Page object's parent Document Object.

```
KlgPfObject
KlgPfPageGetParent(
   KlgPfObject      page
   )
```

KlgPfPageGetType()

Returns the page type.

```
KlgPfPageType
KlgPfPageGetType(
   KlgPfObject      page
   )
```

KlgPfPageGetWidth()

Returns the Page object's width, measured in document units.

```
double
KlgPfPageGetWidth(
   KlgPfObject      page
   )
```

KlgPfPageGetX()

Returns the Page object's `x` offset, measured in document units.

```
double
KlgPfPageGetX(
   KlgPfObject      page
   )
```

KlgPfPageGetY()

Gets the Page object's `y` offset, measured in document units.

```
double
KlgPfPageGetY(
   KlgPfObject      page
   )
```

KlgPfPageSetColor()

Sets the page's background Color object.

```
Boolean
KlgPfPageSetColor(
   KlgPfObject      page,
   KlgPfObject      color
   )
```

KlgPfPageSetFlowPageTemplate()

Sets the template to use when *KlgPfFlowFormFeed()* is called.

```
Boolean
KlgPfPageSetFlowPageTemplate(
   KlgPfObject      page,
   String           page_template
   )
```

Parameters

page: A Page object.

page_template: Name of the Page template.

KlgPfPageSetFlowSectionTemplate()

Sets the Page template to use when *KlgPfFlowSectionBreak()* is called.

```
Boolean
KlgPfPageSetFlowSectionTemplate(
   KlgPfObject      page,
   String           page_template
   )
```

Parameters

page: A Page object.

page_template: Name of the Page template.

KlgPfPageSetHeight()

Sets the page's height.

```
Boolean
KlgPfPageSetHeight(
   KlgPfObject      page,
   double           height
   )
```

KlgPfPageSetLayoutStatus()

Sets page's `LayoutStatus` attribute. This method should be used with care, since it may affect macro evaluation.

```
Boolean
KlgPfPageSetLayoutStatus(
   KlgPfObject          page,
   KlgPfLayoutStatus    status
   )
```

Parameters

page: A Page object.

status: A layout status.

KlgPfPageSetWidth()

Sets the page's width, in document units.

```
Boolean
KlgPfPageSetWidth(
   KlgPfObject      page,
   double           width
   )
```

KlgPfPageSetX()

Sets the page's `X` offset relative to the Document object's upper left corner. The value will be interpreted terms of document units.

```
Boolean
KlgPfPageSetX(
   KlgPfObject      page,
   double           X
   )
```

KlgPfPageSetY()

Sets the page's `Y` offset relative to the Document object's upper left corner. The value will be interpreted terms of document units.

```
Boolean
KlgPfPageSetY(
   KlgPfObject      page,
   double           Y
   )
```

KlgPfPageStringToFrame()

Gets the named Frame object from the Page object's `FrameList`. Returns `NULL` if not found.

```
KlgPfObject
KlgPfPageStringToFrame(
   KlgPfObject      page,
   String           frame_name
   )
```

Parameters

page: A Page object.

frame_name: Name of the frame. PageFormatter is not case sensitive.

KlgPfPageTemplateCreate()

Creates a template Page object. The Page template data can be read in from a named file, from an open input stream, or from a string buffer defined in the code. To read a template in from a named file, the *io_type* parameter should be `KlgPfIOFilename` and the *data* parameter should be the file name. To read a template from an open stream, the *io_type* parameter should be `KlgPfIOStream` and the *data* parameter should be the stream. To create a template from a string, the *io_type* parameter should be `KlgPfIOBuffer` and the *data* parameter should be the string buffer.

```
Boolean
KlgPfPageTemplateCreate(
   KlgPfObject      doc,
   KlgPfIOType      io_type,
   XtPointer        data
   )
```

Parameters

doc: The Page object's Document object.

io_type: Source of the Page template data. May be `KlgPfIOFilename`, `KlgPfIOStream`, or `KlgPfIOBuffer`.

data: A pointer to the Page template data.

K

Printer Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Printer object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

K.1 Object Synopsis

Include File: <KlgPfAll.h> and one or more of <KlgPfPS.h>, <KlgPfAcro.h>, <KlgPfPCL.h> or <KlgPfX11.h>

Description: The Printer object provides attributes and methods that control the "physical" end of printing: the number of pages to output, the number of copies, whether copies should be collated, whether output should be sent to a file instead of paper, and so on.

K.2 Attributes (Name, Type, Default)

Collate **Boolean** **False**

If `NumCopies` is greater than 1, specifies whether the document should be collated.

CurrentDocument **KlgPfObject** **NULL**

The Document object the Printer object is currently printing. When this is set to anything but `NULL`, the Printer object is not usable for output by any other Document object.

DisplayX11 **KlgPfObject** **NULL**

Specifies an X11 screen printer's Display X11 object. This attribute is ignored for all other types of printers.

Filename **String** **NULL**

The file name an Acrobat, PCL or PostScript printer should print to if `OutputType` is set to `KlgPfIOFilename`.

FontFamilyList **KlgPfObject** ***Dynamic***

The list of Font Family objects supported by the printer.

FontList **KlgPfObject** ***Dynamic***

The list of Font objects supported by the printer.

Name **String** ***Dynamic***

The ASCII name of the printer.

NumCopies **int** **1**

The number of copies to print.

OutputCallback **KlgPfCallbackRecord** ***Dynamic***

Specifies a procedure (along with a pointer to any relevant data) that should be called when an output callback is generated. When PageFormatter generates an output callback, it passes the following structure:

```
typedef struct {
   KlgPfCallbackReason   reason;
   KlgPfObject           document;
   KlgPfObject           printer;
} KlgPfOutputCallbackStruct
```

The *reason* member is one of the following values:

`KLGPF_REASON_OUTPUT_NEW`	PageFormatter begins an output process.
`KLGPF_REASON_OUTPUT_PAGE`	PageFormatter outputs a page.
`KLGPF_REASON_OUTPUT_END`	PageFormatter ends an output process.

OutputPageEnd **int** **KLGPF_MAXINT**

When printing a range of pages, `OutputPageEnd` signifies the last page to print in the range. `KLGPF_MAXINT` implies print to the end of the document.

OutputPageStart **int** **0**

When printing a range of pages, `OutputPageStart` signifies the first page to print in the range.

OutputType **KlgPfIOType** **KlgPfIOStream**

Specifies the output method for an Acrobat, PCL or PostScript printer. Valid output methods are:

KlgPfIOStream	Print to the printer.
KlgPfIOFilename	Print to a file defined by the `Filename` attribute.

PaperSizeUnits **KlgPfUnit** **KLGPF_UNIT_INCHES**

Specifies the units that should be used to interpret the values associated with the `PaperSize`. Valid units are:

`KLGPF_UNIT_CM`	Centimeters
`KLGPF_UNIT_INCHES`	Inches
`KLGPF_UNIT_POINTS`	Points

PrinterOptionsList **KlgPfObject** ***Dynamic***

A list of supported PtrOption objects for the current print device.

PrinterOptionSelectedList **KlgPfObject** ***Dynamic***

A list of selected PtrOptionValue objects.

PrinterType **KlgPfPrinterTypeEnum** ***Dynamic***

Specifies the type of printer. Valid types are:

`KLGPF_PRINTER_TYPE_ACROBAT`	Prints to an Adobe Acrobat PDF.
`KLGPF_PRINTER_TYPE_PS`	Prints to a PostScript printer.
`KLGPF_PRINTER_TYPE_PCL`	Prints to a PCL printer.
`KLGPF_PRINTER_TYPE_X11`	Prints to an X11 Window printer.

ReverseOrder **Boolean** **False**

When set to `True`, the document will be printed in reverse order (that is, from back to front).

Stream **FILE *** **NULL**

If `OutputType` is set to `KlgPfIOStream`, `stream` specifies an open output stream. This is only used by Acrobat, PCL and PostScript Printer objects.

Type **XtPointer** ***Dynamic***

If the printer is created with *KlgPfPrinterCreatePS(),* `Type` is a handle to a PostScript printer structure. If the printer is created with *KlgPfPrinterCreateAcro(),* `Type` is a

handle to an Acrobat printer structure. If the printer is created with *KlgPfPrinterCreatePCL()*, `Type` is a handle to a PCL printer structure. If the printer is created with *KlgPfPrinterCreateX11()*, `Type` is a handle to an X11 Window printer structure.

XResolution **double** **75**

The horizontal (X) printer resolution settings. By default, XResolution is 75 or 100 depending on the printer output device's resolution.

YResolution **double** **75**

The vertical (Y) printer resolution settings. By default, YResolution is 75 or 100 depending on the printer output device's resolution.

K.3 Methods

KlgPfObjectIsPrinter()

Returns `True` if the object is a Printer object.

```
Boolean
KlgPfObjectIsPrinter(
   KlgPfObject   object
   )
```

KlgPfPrinterClearSelectedOptions()

Removes all PtrOptionsValue objects from the `PrinterOptionSelectedList` list.

```
Boolean
KlgPfPrinterClearSelectedOptions(
   KlgPfObject   printer
   )
```

KlgPfPrinterCreatePCL()

Creates a PCL Printer object.

```
KlgPfObject
KlgPfPrinterCreatePCL(
   String        name
   )
```

Returns

Returns a Printer object.

Parameters

name: a string name. This name is not used to define the destination PCL printer device.

KlgPfPrinterCreatePS()

Creates a PostScript Printer object.

```
KlgPfObject
KlgPfPrinterCreatePS(
   String        name
   )
```

Returns

Returns a Printer object.

Parameters

name: a string name. This name is not used to define the destination PostScript printer device.

KlgPfPrinterCreateX11()

Creates an X window Printer object.

```
KlgPfObject
KlgPfPrinterCreateX11(
   KlgPfObject   displayX11
   )
```

Returns

Returns a Printer object.

Parameters

display: A Display X11 object.

KlgPfPrinterDestroy()

Destroys the Printer object.

```
void
KlgPfPrinterDestroy(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetCollate()

Returns `True` if set to collate multiple copies.

```
Boolean
KlgPfPrinterGetCollate(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetCurrentDocument()

Returns the Document object the Printer object is currently using to render output.

```
KlgPfObject
KlgPfPrinterGetCurrentDocument(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetDisplayX11()

Returns the Display X11 object the X11 Printer is currently using. This method is only used for X11 printers.

```
KlgPfObject
KlgPfPrinterGetDisplayX11(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetFilename()

Returns the name of the file to which the Printer object is printing. This method is only used for PostScript printers.

```
String
KlgPfPrinterGetFilename(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetFontFamilyList()

Returns the list of available Font Family objects.

```
KlgPfObject
KlgPfPrinterGetFontFamilyList(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetFontList()

Returns the list of Font objects the printer supports.

```
KlgPfObject
KlgPfPrinterGetFontList(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetName()

Returns the ASCII name of the printer.

```
String
KlgPfPrinterGetName(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetNumCopies()

Returns the number of copies setting.

```
int
KlgPfPrinterGetNumCopies(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetOptionValuesByOptionName()

Returns a PtrOptionValues list from the `PrinterOptionsList` list by specifying the parent PtrOption object's `Name` attribute (for example, "PageSize"). This method is only used for PostScript printers.

```
KlgPfObject
KlgPfPrinterGetOptionValuesByOptionName(
   KlgPfObject   printer,
   String        name
   )
```

KlgPfPrinterGetOutputPageEnd()

Returns the last page number to be printed or `KLGPF_MAXINT` if none is set.

```
int
KlgPfPrinterGetOutputPageEnd(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetOutputPageStart()

Returns the initial page number to be printed.

```
int
KlgPfPrinterGetOutputPageStart(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetOutputType()

Returns the output method. This method is only used for PostScript printers.

```
KlgPfIOType
KlgPfPrinterGetOutputType(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetPrinterOptionsList()

Returns the `PrinterOptionsList` list. This method is only used for PostScript printers.

```
KlgPfObject
KlgPfPrinterGetPrinterOptionsList(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetPrinterType()

Returns the value of the `PrinterType` attribute (for example, `KLGPF_PRINTER_TYPE_PS`).

```
KlgPfPrinterTypeEnum
KlgPfPrinterGetPrinterType(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetReverseOrder()

Returns `True` if the Printer object is set for reverse order printing.

```
Boolean
KlgPfPrinterGetReverseOrder(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetSelectedOptionsList()

Returns a list of PtrOptionValue objects. This method is only used for PostScript printers.

```
KlgPfObject
KlgPfPrinterGetSelectedOptionsList(
   KlgPfObject   printer
)
```

KlgPfPrinterGetResolutionX()

Returns the `XResolution` attribute, specifying horizontal (X) printer resolution settings.

```
double
KlgPfPrinterGetResolutionX(
   KlgPfObject   printer_object
)
```

KlgPfPrinterGetResolutionY()

Returns the `YResolution` attribute, specifying vertical (Y) printer resolution settings.

```
double
KlgPfPrinterGetResolutionY(
   KlgPfObject   printer_object
)
```

KlgPfPrinterGetStream()

Returns the stream accepting output if `OutputType` is set to `KlgPfIOStream`. This method is only used for PostScript printers.

```
FILE *
KlgPfPrinterGetStream(
   KlgPfObject   printer
   )
```

KlgPfPrinterGetType()

Returns the printer type.

```
KlgPfPrinterType
KlgPfPrinterGetType(
   KlgPfObject   printer
   )
```

KlgPfPrinterOutput()

Prints the specified Document object.

```
Boolean
KlgPfPrinterOutput(
   KlgPfObject   printer,
   KlgPfObject   document
   )
```

KlgPfPrinterSetCollate()

Sets the Printer object's `Collate` attribute.

```
Boolean
KlgPfPrinterSetCollate(
   KlgPfObject   printer,
   Boolean       collate
   )
```

Parameters

printer: A Printer object.

collate: `True` enables collating. `False` disables collating.

KlgPfPrinterSetDisplayX11()

Sets the Printer object's `DisplayX11` attribute to the specified Display X11 object. This method is only used for X11 printers.

```
Boolean
KlgPfPrinterSetDisplayX11(
   KlgPfObject   printer,
   KlgPfObject   displayX11_obj
   )
```

KlgPfPrinterSetFilename()

Sets the file name to use if `OutputType` is set to `KlgPfIOFilename`. This method is only used for PostScript printers.

```
Boolean
KlgPfPrinterSetFilename(
   KlgPfObject   printer,
   String        filename
```

KlgPfPrinterSetNumCopies()

Sets the number of copies to print.

```
Boolean
KlgPfPrinterSetNumCopies(
   KlgPfObject   printer,
   int           num_copies
   )
```

KlgPfPrinterSetOutputCallback()

Sets a procedure to call (along with a pointer to any relevant data) when PageFormatter generates an output callback.

```
Boolean
KlgPfPrinterSetOutputCallback(
   KlgPfObject          printer,
   KlgPfCallbackProc    callback,
   XtPointer            call_data
   )
```

Parameters

printer: The Printer object

callback: The procedure to call. The procedure must be defined to be the following type:

```
typedef void (*KlgPfCallbackProc)(
   XtPointer        cbs,
   XtPointer        call_data
   );
```

call_data: A handle to data.

KlgPfPrinterSetOutputPageEnd()

Sets the last page in a range to print. You can pass this method `KLGPF_MAXINT` if you want the printer to print to the end of the document.

```
Boolean
KlgPfPrinterSetOutputPageEnd(
   KlgPfObject   printer,
   int           end_page
   )
```

KlgPfPrinterSetOutputPageStart()

Sets the first page in a range to print.

```
Boolean
KlgPfPrinterSetOutputPageStart(
   KlgPfObject   printer,
   int           start_page
   )
```

KlgPfPrinterSetOutputType()

Sets the output type (for example, *KlgPfIOFilename*).

```
Boolean
KlgPfPrinterSetOutputType(
   KlgPfObject   printer,
   KlgPfIOType   type
   )
```

KlgPfPrinterSetPrinterOption()

Adds a PtrOptionValue object to the `PrinterOptionSelectedList` list.

```
Boolean
KlgPfPrinterSetPrinterOption(
   KlgPfObject   printer,
   KlgPfObject   option_value
   )
```

KlgPfPrinterSetReverseOrder()

Sets the print order to forward or reverse. When set to `True`, the document will be printed in reverse order (that is, from back to front)

```
Boolean
KlgPfPrinterSetReverseOrder(
   KlgPfObject   printer,
   Boolean       reverse
   )
```

KlgPfPrinterSetStream()

If `OutputType` is set to `KlgPfIOStream`, sets an open output stream. This method is only used for Acrobat, PCL and PostScript printers.

```
Boolean
KlgPfPrinterSetStream(
   KlgPfObject   printer,
   FILE          *stream
   )
```

KlgPfPrinterStringToFontFamily()

Gets a Font Family object by name from Printer object's font family list.

```
KlgPfObject
KlgPfPrinterStringToFontFamily(
   KlgPfObject   printer,
   String        font_family
   )
```

L

Print Dialog and Preview Widgets

Print Dialog ■ *Preview Widget*

L.1 Print Dialog

L.1.1 Print Dialog Synopsis

Include File: <KlgPfXmPrintDialog.h>

Description: The Print Dialog widget provides methods for creating a Motif widget that lets you provide the end user with a GUI printer dialog box. The Print Dialog's controls correspond to the Printer object's setable attributes. PageFormatter provides a list of convenience methods for controlling the print dialog. Source code for the Print Dialog is provided in *$KLGPRHOME/src/pf/util*.

L.1.2 Print Dialog Methods

KlgPfXmPrintDialogCreate()

Creates a Print dialog widget. Dialog controls are set based on corresponding Printer object attributes.

```
Widget
KlgPfXmPrintDialogCreate(
   Widget        parent,
   KlgPfObject   printer,
   KlgPfObject   document
   )
```

Parameters

parent: A parent widget.

printer: A Printer object.

document: A Document object.

KlgPfXmPrintDialogPopdown()

Hides and/or destroys the Print Dialog object.

```
void
KlgPfXmPrintDialogPopdown(
   Widget        dialog,
   Boolean       destroy
   )
```

Parameters

dialog: The Dialog widget.

destroy: `True` specifies the Dialog widget that should be destroyed

KlgPfXmPrintDialogPopup()

Calls *KlgPfXmPrintDialogUpdate()* and displays the Dialog widget.

```
void
KlgPfXmPrintDialogPopup(
   Widget        dialog,
   KlgPfObject   printer,
   KlgPfObject   document
   )
```

KlgPfXmPrintDialogUpdate()

Updates the Print Dialog's controls to correspond with the Printer object's attributes.

```
Boolean
KlgPfXmPrintDialogUpdate(
   Widget        dialog,
   KlgPfObject   printer,
   KlgPfObject   document
   )
```

L.2 Preview Widget

L.2.1 Preview Widget Synopsis

Include File: <KlgPfXmPreview.h>

Parent Object: None

Description: The Print Preview widget along with its associated methods and attributes allow you to create a document previewer for Motif. Source code for the Preview widget is provided in *$KLGPRHOME/src/pf/util*.

L.2.2 Attributes (Name, Type, Default)

CurrentPage int 1

Specifies the current page displayed.

Document **KlgPfObject** **NULL**

Specifies the Document object being previewed.

Height **int** **NULL**

Specifies the previewer widget's height on creation. You may only set this attribute at create time.

Printer **KlgPfObject** **NULL**

Specifies the Printer object used to render the document.

Width **int** **NULL**

Specifies the previewer widget's width on creation. You may only set this attribute at create time.

L.2.3 Methods

KlgPfXmPreviewCreate()

Creates and returns a Print Preview widget. If you pass *KlgPfXmPreviewCreate()* zero `height` and `width` parameters, PageFormatter uses default settings.

```
Widget
KlgPfXmPreviewCreate(
   Widget        parent,
   int           width,
   int           height
   )
```

Parameters

parent: A parent widget.

width: The widget's width.

height: The widget's height.

KlgPfXmPreviewGetPageNumber()

Returns the page number of the page being displayed.

```
int
KlgPfXmPreviewGetPageNumber(
   Widget        preview
   )
```

KlgPfXmPreviewGetDocument()

Gets the Document object being previewed.

```
KlgPfObject
KlgPfXmPreviewGetDocument(
   Widget        preview
   )
```

KlgPfXmPreviewGetPrinter()

Gets the Printer object used to render the Document object.

```
KlgPfObject
KlgPfXmPreviewGetPrinter(
   Widget        preview
   )
```

KlgPfXmPreviewPopdown()

Hides or destroys the Print Preview widget.

```
Boolean
KlgPfXmPreviewPopdown(
   Widget        preview,
   Boolean       destroy
   )
```

Parameters

preview: The Print Preview widget.

destroy: `True` specifies the widget should be destroyed.

KlgPfXmPreviewPopup()

Calls *KlgPfXmPreviewUpdate()* and displays the Print Preview widget.

```
void
KlgPfXmPreviewPopup(
   Widget        preview
   )
```

KlgPfXmPreviewSetDocument()

Sets the Document object to preview and calls *KlgPfXmPreviewUpdate()*.

```
Boolean
KlgPfXmPreviewSetDocument(
   Widget        preview,
   KlgPfObject   document
   )
```

KlgPfXmPreviewSetPageNumber()

Sets the `CurrentPage` attribute.

```
Boolean
KlgPfXmPreviewSetPageNumber(
   Widget        preview,
   int           page
   )
```

KlgPfXmPreviewUpdate()

Updates the Print Preview widget and the page displayed based on `CurrentPage` attribute.

```
Boolean
KlgPfXmPreviewUpdate(
   Widget        preview
   )
```

M

PtrOption and PtrOptionValue Objects

PtrOption Object Synopsis ■ *PtrOption Object Methods*

This appendix lists all the PtrOption and PtrOptionValue objects attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

An example of how to use these objects is found in the X11 previewer code in *$KLGPRHOME/src/pf/util*.

M.1 PtrOption Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfPtrOption.h>

Parent Object: Printer Object

Usage: `PrinterOptionList`

Description: The PtrOption object provides attributes and methods for defining printer options (for example, PageSize, Resolution).

M.1.1 PtrOption Object Attributes (Name, Type, Default)

DefaultValue **KlgPfObject** ***Dynamic***

The default PtrOptionValue object, if a default exists in the printer definition. If there is no default, this value is `NULL`.

Description **String** ***Dynamic***

Descriptive text identifying the PtrOption object.

Name **String** ***Dynamic***

ASCII Name of the PtrOption object (for example “PageSize”).

ValueList **KlgPfObject** ***Dynamic***

A list of possible PtrOptionValue objects. For example, a PageSize PtrOption object's `ValueList` list could contain a *Letter* PtrOptionValue object, a *Legal* PtrOptionValue object, an *A4* PtrOptionValue object, and so on.

M.1.2 PtrOption Object Methods

KlgPfObjectIsPtrOption()

Returns `True` if the object is an PtrOption object, otherwise `False`.

```
Boolean
KlgPfObjectIsPtrOption(
   KlgPfObject   ptr_option
   )
```

KlgPfPtrOptionCreate()

Creates and returns an PtrOption object.

```
KlgPfObject
KlgPfPtrOptionCreate(
   KlgPfObject   parent,
   String        name,
   String        description
   )
```

Parameters

parent: The Printer object

name: The PtrOption object's ASCII name (for example, "PageSize").

description: Longer, descriptive ASCII text.

KlgPfPtrOptionDestroy()

Destroys an PtrOption object. Note: destroying the parent object automatically destroys this object. You should not call this method if you have destroyed the parent object.

```
void
KlgPfPtrOptionDestroy(
   KlgPfObject   ptr_option
   )
```

KlgPfPtrOptionFindOptionByName()

Finds and returns an PtrOption object on a list (for example, its parent `PrinterOptionList`).

```
KlgPfObject
KlgPfPtrOptionFindOptionByName(
   KlgPfObject   option_list,
   String        name
   )
```

Parameters

option_list: A List object containing PtrOption objects.

name: The ASCII `Name` attribute of the desired PtrOption object.

KlgPfPtrOptionGetDefaultValue()

Returns the default PtrOptionValue object or `NULL` if one is not defined in the printer definition (that is, the PPD file).

```
KlgPfObject
KlgPfPtrOptionGetDefaultValue(
   KlgPfObject   ptr_option
   )
```

KlgPfPtrOptionGetDescription()

Returns the ASCII `Description` attribute of the specified PtrOption object.

```
String
KlgPfPtrOptionGetDescription(
   KlgPfObject   ptr_option
   )
```

KlgPfPtrOptionGetName()

Returns the ASCII `Name` attribute of the specified PtrOption object.

```
String
KlgPfPtrOptionGetName(
   KlgPfObject   ptr_option
)
```

KlgPfPtrOptionGetOptionValuesByName()

Finds an PtrOption object on a list by its `Name` attribute and returns its `ValueList` list of PtrOptionValue objects.

```
KlgPfObject
KlgPfPtrOptionGetOptionValuesByName(
   KlgPfObject   option_list,
   String        name
   )
```

Parameters

option_list: A List object containing PtrOption objects.

name: The ASCII `Name` attribute of the desired PtrOption object.

KlgPfPtrOptionGetValuesList()

Returns a list of PtrOptionValue objects.

```
KlgPfObject
KlgPfPtrOptionGetValuesList(
   KlgPfObject   ptr_option
   )
```

KlgPfPtrOptionSetDefaultValue()

Sets the PtrOption object's `DefaultValue` attribute to the specified PtrOptionValue object.

```
Boolean
KlgPfPtrOptionSetDefaultValue(
   KlgPfObject   ptr_option,
   KlgPfObject   option_value
   )
```

M.2 PtrOptionValue Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfPtrOptVal.h>

Parent Object: PtrOption object

Usage: `DefaultValue`, `ValueList`

Description: The PtrOptionValue object provides attributes and methods for defining individual values available for each PtrOption object.

M.2.1 PtrOptionValue Attributes (Name, Type, Default)

Description **String** ***Dynamic***

ASCII description of the option value.

Name **String** ***Dynamic***

ASCII name of the option value.

M.2.2 PtrOptionValue Methods

KlgPfObjectIsPtrOptVal()

Returns `True` if the object is an PtrOptionValue object, otherwise `False`.

```
Boolean
KlgPfObjectIsPtrOptVal(
   KlgPfObject   printer_option_value
   )
```

KlgPfPtrOptValCreate()

Creates and returns an PtrOptionValue object.

```
KlgPfObject
KlgPfPtrOptValCreate(
   KlgPfObject   parent,
   String        name,
   String        description
   )
```

Parameters

parent: The PtrOption object

name: The PtrOption object's ASCII name (for example, "Legal").

description: Longer, descriptive ASCII text.

KlgPfPtrOptValDestroy()

Destroys an PtrOptionValue object. Note: destroying the parent object automatically destroys this object. You should not call this method if you have destroyed the parent object.

```
void
KlgPfPtrOptValDestroy(
   KlgPfObject   printer_option_value
   )
```

KlgPfPtrOptValFindByName()

Finds and returns a PtrOptionValue object matching *Name* in the given list (usually its parent PtrOption's `ValueList`).

```
KlgPfObject
KlgPfPtrOptValFindByName(
   KlgPfObject       value_list,
   String                name
```

KlgPfPtrOptValGetDescription()

Returns the PtrOptionValue object's ASCII description.

```
String
KlgPfPtrOptValGetDescription(
   KlgPfObject   printer_option_value
   )
```

KlgPfPtrOptValGetName()

Returns the PtrOptionValue object's ASCII name.

```
String
KlgPfPtrOptValGetName(
   KlgPfObject   printer_option_value
   )
```

KlgPfPtrOptValGetOptionName()

Returns the PtrOptionValue object's parent PtrOption object.

```
String
KlgPfPtrOptValGetOptionName(
   KlgPfObject   printer_option_value
   )
```

N

Style Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Style object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

N.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfStyle.h>

Parent Object: Document Object

Usage: `CurrentStyle`, `StyleList`

Description: The Style object's attributes dictate how the rendered text should appear. Attributes like font, indents, and line spacing are defined by the Style object.

N.2 Attributes (Name, Type, Default)

Alignment KlgPfParagraphAlignment KLGPF_ALIGN_LEFT

The paragraph alignment attribute. Valid alignments are:

`KLGPF_USING_TABS`	For PageFormatter use only. Do not use.
`KLGPF_ALIGN_LEFT`	Paragraph text is left aligned.
`KLGPF_ALIGN_CENTER`	Paragraph text is centered. You cannot use tabs in a centered paragraph.
`KLGPF_ALIGN_RIGHT`	Paragraph text is right aligned. You cannot use tabs in a right-aligned paragraph.

KLGPF_JUSTIFY	Paragraph text is left and right justified. You cannot use tabs in a justified paragraph.

Color **KlgPfObject** **"black"**

Color of the text to be rendered.

Font **KlgPfObject** **"Times-Roman"**

The specific font to be used to print text (for example "Times-Roman").

FontFamily **KlgPfObject** **"Times"**

Font family (for example, "Times") to be used when rendering text.

FontSize **int** **12**

Font size, in points, to be used when rendering text.

FontStyle **KlgPfFontStyle** **KLGPF_FONT_NORMAL**

Font style to be used when rendering text. Valid font styles are:

KLGPF_FONT_NORMAL	Renders the text in its normal face.
KLGPF_FONT_BOLD	Renders the text in bold.
KLGPF_FONT_ITALIC	Renders the text in italics.
KLGPF_FONT_BOLDITALIC	Renders the text in bold and italic.

IndentFirstLine **double** **0.0**

The amount, in document units, the text in the first line of a paragraph is rendered from the left side of the current column. This attribute affects only the first line. Subsequent lines will be rendered based on the `IndentLeft` attribute.

IndentLeft **double** **0.0**

The amount, in document units, paragraph text is rendered from the left side of the current column. This attribute affects all lines in the paragraph, except the first line.

IndentRight **double** **0.0**

The amount, in document units, paragraph text is rendered from the right side of the frame or column. This attribute affects all lines in the paragraph, including the first line.

LineSpacing **double** **0.05**

The amount, in document units, paragraph lines are spaced apart.

Name **String** ***Dynamic***

Name of the style. For example, “Heading 1”.

ParagraphSpacing **double** **0.1**

The amount of space, in document units, to leave after a paragraph break.

SubscriptRatio **double** **0.75**

Size ratio of the superscript/subscript font to the “regular” font size as defined by the `FontSize` attribute. For example, a 12 point `FontSize` and a 0.75 `SubscriptRatio` will produce a 9 point superscript/subscript font.

TabList **KlgPfObject** ***Dynamic***

A list of tab stops to be used on a line of text. Tabs are kept in sorted order.

TextPosition **KlgPfTextPosition** **KLGPF_FONT_ON_BASELINE**

Specifies whether the text should be rendered as regular “baseline” text, a subscript, or a superscript. Valid text positions are:

`KLGPF_FONT_ON_BASELINE`	Renders text in the normal fashion, along the baseline.
`KLGPF_FONT_SUBSCRIPT`	Renders below the baseline based on the `SubscriptRatio` attribute.
`KLGPF_FONT_SUPERSCRIPT`	Renders above the baseline based on the `SubscriptRatio` attribute.

Underline **Boolean** **False**

Flag indicating whether to render text with an underscore. PageFormatter places underlining at -10% of the font height relative to the baseline. The thickness of the underline is approximately 5% of font height.

N.3 Methods

KlgPfObjectIsStyle()

Returns `True` if the object is a Style object, otherwise `False`.

```
Boolean
KlgPfObjectIsStyle(
   KlgPfObject   object
   )
```

KlgPfStyleClearTabs()

Destroys all Tab objects in the specified Style object.

```
Boolean
KlgPfStyleClearTabs(
   KlgPfObject   style
   )
```

Returns

Returns `True` if the Tab objects have been destroyed successfully.

KlgPfStyleCopy()

Returns a copy of the specified Style object. If the passed *name* already exists in the `StyleList`, `NULL` is returned.

```
KlgPfObject
KlgPfStyleCopy(
   KlgPfObject   source,
   KlgPfObject   parent,
   String        name
   )
```

Parameters

source: The Style object you wish to copy.

parent: The new Style object's parent Document object.

name: Name of the new Style object. For example, "Heading 1B".

KlgPfStyleCreate()

Creates and returns a Style object, initializes its attributes with default values, and adds the style to the specified Document object's `StyleList`. Note, if the passed *name* already exists in the `StyleList` in a given document, the previously created style with that name is returned instead.

```
KlgPfObject
KlgPfStyleCreate(
   KlgPfObject   parent,
   String        name
   )
```

Parameters

parent: The Style object's parent Document object.

name: A style name. For example, "Heading 1".

KlgPfStyleDestroy()

Destroys a Style object. Note: destroying the parent object automatically destroys this object. You should not call this method if you have destroyed the parent object.

```
void
KlgPfStyleDestroy(
   KlgPfObject   style
   )
```

KlgPfStyleGetAlignment()

Returns the paragraph alignment (for example `KLGPF_ALIGN_LEFT`).

```
KlgPfParagraphAlignment
KlgPfStyleGetAlignment(
   KlgPfObject   style
   )
```

KlgPfStyleGetColor()

Returns the Color object applied to the rendered text.

```
KlgPfObject
KlgPfStyleGetColor(
   KlgPfObject   style
   )
```

KlgPfStyleGetFont()

Returns the Font object being used to render the text.

```
KlgPfObject
KlgPfStyleGetFont(
   KlgPfObject   style
   )
```

KlgPfStyleGetFontFamily()

Returns the current Font Family object.

```
KlgPfObject
KlgPfStyleGetFontFamily(
   KlgPfObject   style
   )
```

KlgPfStyleGetFontSize()

Returns the point size used to render the font.

```
int
KlgPfStyleGetFontSize(
   KlgPfObject   style
   )
```

KlgPfStyleGetFontStyle()

Gets current font style (for example `KLGPF_FONT_BOLD`).

```
KlgPfFontStyle
KlgPfStyleGetFontStyle(
   KlgPfObject   style
   )
```

KlgPfStyleGetIndentFirstLine()

Returns the style's left-indent setting for the first line of the paragraph (that is, the position, in document units, from the left side of the current column).

```
double
KlgPfStyleGetIndentFirstLine(
   KlgPfObject   style
   )
```

KlgPfStyleGetIndentLeft()

Returns the style's left-indent setting for subsequent lines of the paragraph (that is, the position, in document units, from the left side of the current column).

```
double
KlgPfStyleGetIndentLeft(
   KlgPfObject   style
   )
```

KlgPfStyleGetIndentRight()

Returns the style's right-indent setting for all lines of the paragraph (that is, the position, in document units, from the right side of the current column).

```
double
KlgPfStyleGetIndentRight(
   KlgPfObject   style
   )
```

KlgPfStyleGetLineSpacing()

Returns amount of inter-paragraph spacing, in document units, added after a paragraph break.

```
double
KlgPfStyleGetLineSpacing(
   KlgPfObject   style
   )
```

KlgPfStyleGetName()

Returns the Style object's ASCII name.

```
String
KlgPfStyleGetName(
   KlgPfObject   style
   )
```

KlgPfStyleGetParagraphSpacing()

Returns the size ratio of the superscript/subscript font to the "regular" font size.

```
double
KlgPfStyleGetParagraphSpacing(
   KlgPfObject   style
   )
```

KlgPfStyleGetSubscriptRatio()

Returns the ratio of the subscript/superscript font size to the base font size.

```
double
KlgPfStyleGetSubscriptRatio(
   KlgPfObject   style
   )
```

KlgPfStyleGetTabList()

Returns the list of Tab objects associated with the Style object.

```
KlgPfObject
KlgPfStyleGetTabList(
   KlgPfObject   style
   )
```

KlgPfStyleGetTextPosition()

Returns the `KlgPfTextPosition` type identifying if the text is rendered as a subscript (`KLGPF_FONT_SUBSCRIPT`), a superscript (`KLGPF_FONT_SUPERSCRIPT`), or on the baseline (`KLGPF_FONT_ON_BASELINE`).

```
KlgPfTextPosition
KlgPfStyleGetTextPosition(
   KlgPfObject   style
   )
```

KlgPfStyleGetUnderline()

Returns `True` if underlining is enabled.

```
Boolean
KlgPfStyleGetUnderline(
   KlgPfObject   style
   )
```

KlgPfStyleSetAlignment()

Sets the paragraph alignment (for example, `KLGPF_ALIGN_LEFT`).

```
Boolean
KlgPfStyleSetAlignment(
   KlgPfObject               style,
   KlgPfParagraphAlignment   alignment
   )
```

KlgPfStyleSetColor()

Sets the Color object that should be applied to rendered text.

```
Boolean
KlgPfStyleSetColor(
   KlgPfObject   style,
   KlgPfObject   color
   )
```

KlgPfStyleSetFont()

Sets the Style object's `font` attribute. Unless a special font not appearing in the standard Font Family list is wanted, font selection by *KlgPfStyleSetFontFamily()* and *KlgPfStyleSetFontStyle()* is preferred.

```
Boolean
KlgPfStyleSetFont(
   KlgPfObject   style,
   KlgPfObject   font
   )
```

KlgPfStyleSetFontFamily()

Sets the Style object's Font Family. This will also set the `Font` attribute based on the `FontFamily` and `FontStyle` attributes.

```
Boolean
KlgPfStyleSetFontFamily(
   KlgPfObject   style,
   KlgPfObject   font_family
   )
```

KlgPfStyleSetFontSize()

Sets the font size in points.

```
Boolean
KlgPfStyleSetFontSize(
   KlgPfObject   style,
   int           font_size
   )
```

KlgPfStyleSetFontStyle()

Sets the Style object's `FontStyle` attribute (for example, `KLGPF_FONT_ITALIC`). This will also set the `Font` attribute based on the `FontFamily` and `FontStyle` attributes.

```
Boolean
KlgPfStyleSetFontStyle(
   KlgPfObject   style,
   KlgPfFontStylefont_style
   )
```

KlgPfStyleSetIndentFirstLine()

Sets the amount, in document units, the text in the first line of a paragraph is rendered from the left side of the current column.

```
Boolean
KlgPfStyleSetIndentFirstLine(
   KlgPfObject   style,
   double        indent_first_line
   )
```

KlgPfStyleSetIndentLeft()

Sets the amount, in document units, paragraph lines subsequent to the first paragraph line are rendered from the left side of the current column.

```
Boolean
KlgPfStyleSetIndentLeft(
   KlgPfObject   style,
   double        indent_left
   )
```

KlgPfStyleSetIndentRight()

Sets the right indent amount, in document units, paragraph text is rendered from the right side of the current column.

```
Boolean
KlgPfStyleSetIndentRight(
   KlgPfObject   style,
   double        indent_right
   )
```

KlgPfStyleSetLineSpacing()

Sets the amount, in document units, paragraph lines are spaced apart.

```
Boolean
KlgPfStyleSetLineSpacing(
   KlgPfObject   style,
   double        spacing
   )
```

KlgPfStyleSetParagraphSpacing()

Sets the amount of space, in document units, to leave after a paragraph break.

```
Boolean
KlgPfStyleSetParagraphSpacing(
   KlgPfObject   style,
   double        spacing
   )
```

KlgPfStyleSetSubscriptRatio()

Sets the ratio of the superscript/subscript font to the "main" font as defined by the `FontSize` attribute.

```
Boolean
KlgPfStyleSetSubscriptRatio(
   KlgPfObject   style,
   double        subscript_ratio
   )
```

KlgPfStyleSetTabsByValueList()

Lets you create a set of Tab objects and set their x attributes from an array of values.

```
Boolean
KlgPfStyleSetTabsByValueList(
   KlgPfObject   style,
   double        *tab_values,
   int           ntabs
   )
```

Parameters

style: The Style object.

tab_values: An array of x values. PageFormatter assumes there are at least *ntabs* values in the tab values array.

ntabs: The number of Tab objects to create.

KlgPfStyleSetTabsEquallySpaced()

Creates a series of Tab objects, spaced at equal intervals.

```
Boolean
KlgPfStyleSetTabsEquallySpaced(
   KlgPfObject   style,
   double        start_value,
   double        increment_value,
   int           ntabs
   )
```

Parameters

style: The Style object.

start_value: The x position of the first Tab object.

increment_value: How far apart the Tab stops should be set.

ntabs: The number of Tab objects to create.

KlgPfStyleSetTextPosition()

Sets text to render as a subscript (KLGPF_FONT_SUBSCRIPT), a superscript (KLGPF_FONT_SUPERSCRIPT), or on the baseline (KLGPF_FONT_ON_BASELINE).

```
Boolean
KlgPfStyleSetTextPosition(
   KlgPfObject          style,
   KlgPfTextPosition    position
   )
```

KlgPfStyleSetUnderline()

Specify whether or not to render text with an underline.

```
Boolean
KlgPfStyleSetUnderline(
   KlgPfObject   style,
   Boolean       underline
   )
```

Parameters

style: The Style object.

underline: `True` renders text with an underline. `False` renders text without an underline.

0

Tab Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Tab object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

0.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfTab.h>

Parent Object: Style Object

Usage: `TabList`

Description: The Tab object provides methods for adding tabs to the Style object's `TabList` attribute and modifying the tab's type, horizontal position, and appearance (that is, its fill type).

0.2 Attributes (Name, Type, Default)

FillType **KlgPfFillType** **KLGPF_TAB_NO_FILL**

Character pattern used to fill the white space after the tab. Valid fill types are:

`KLGPF_TAB_NO_FILL`	No fill. Blank space.
`KLGPF_TAB_FILL_WITH_UNDERSCORE`	Fill with underscore ("_") characters.
`KLGPF_TAB_FILL_WITH_ELLIPSES`	Fill with period ("...") characters.

TabType KlgPfTabType KLGPF_TAB_LEFT

The type of tab stop set. Valid fill types are:

KLGPF_TAB_RIGHT	Characters following the tab are right aligned at the point defined by the X attribute.
KLGPF_TAB_LEFT	Characters following the tab are left aligned at the point defined by the X attribute.
KLGPF_TAB_CENTER	Characters following the tab are centered at the point defined by the X attribute.
KLGPF_TAB_DECIMAL	Characters following the tab are aligned at their decimal place at the point defined by the X attribute.

X double previous tab value + 0.5 document units

The tab "stop", in document units, from the left hand side of the column.

0.3 Methods

KlgPfObjectIsTab()

Returns True if the object is a Tab object, otherwise False.

```
Boolean
KlgPfObjectIsTab(
   KlgPfObject   object
   )
```

KlgPfTabCopy()

Returns a copy of the specified Tab object.

```
KlgPfObject
KlgPfTabCopy(
   KlgPfObject   source,
   KlgPfObject   parent
   )
```

Parameters

source: The Tab object you wish to copy.

parent: The new Tab object's parent Style object.

KlgPfTabCreate()

Adds a tab stop to the indicated Style object's TabList. By default, this will create a left-aligned tab with no fill attribute. The tab is inserted in sorted order (sorted by X attribute).

```
KlgPfObject
KlgPfTabCreate(
   KlgPfObject   parent,
   double        x
   )
```

Parameters

parent: The Tab object's parent Style object.

x: The tab stop in document units.

KlgPfTabDestroy()

Destroys the Tab object. Note: destroying the parent object automatically destroys this object. You should not call this method if you have destroyed the parent object.

```
void
KlgPfTabDestroy(
   KlgPfObject   tab_object
   )
```

KlgPfTabGetFillType()

Returns the character pattern type (for example, `KLGPF_TAB_FILL_WITH_ELLIPSES`) used to fill the tab's leading white space.

```
KlgPfFillType
KlgPfTabGetFillType(
   KlgPfObject   tab_object
   )
```

KlgPfTabGetTabType()

Returns the Tab object's `type` (for example, `KLGPF_TAB_LEFT`).

```
KlgPfTabType
KlgPfTabGetTabType(
   KlgPfObject   tab_object
   )
```

KlgPfTabGetX()

Returns the tab stop, in document units, from the left hand side of the frame or column.

```
double
KlgPfTabGetX(
   KlgPfObject   tab_object
   )
```

KlgPfTabSetFillType()

Sets the character pattern (for example, `KLGPF_TAB_FILL_WITH_ELLIPSES`) used to fill the tab's leading white space.

```
Boolean
KlgPfTabSetFillType(
   KlgPfObject   tab_object,
   KlgPfFillType fill_type
   )
```

KlgPfTabSetTabType()

Sets the tab type (for example, KLGPF_TAB_RIGHT).

```
Boolean
KlgPfTabSetTabType(
   KlgPfObject   tab_object,
   KlgPfTabType  tab_type
   )
```

KlgPfTabSetX()

Sets the tab stop, in document units, from the left hand side of the frame or column.

```
Boolean
KlgPfTabSetX(
   KlgPfObject   tab_object,
   double        x
   )
```

P

Table Object

Object Synopsis ■ *Attributes (Name, Type, Default)* ■ *Methods*

This appendix lists all the Table object attributes and methods in alphabetical order. Listed after an attribute is its data type and default value.

P.1 Object Synopsis

Include File: <KlgPfAll.h> or <KlgPfTable.h>

Parent Object: Document Object or Frame Object

Usage: `TableList`

Description: The Table object and its sub-objects provide methods for creating and defining a table. **Note:** You must destroy all Table objects you create.

P.2 Attributes (Name, Type, Default)

The Table object has as child objects the Border object, the Row object, and the Column object. The Row and Column objects have Border objects and Cell objects as child objects. The objects are listed here in alphabetical order.

Border

The Border object's attributes define a border that can be placed at the edge of a Cell object, a Row object, a Column object, or a Table object.

Color **KlgPfObject** **Black**

Color object defining the border color.

RuleType **KlgPfTableRuleType** **KLGPF_TABLE_RULE_SINGLE**

Specifies what kind of ruling line the border should render.

KLGPF_TABLE_RULE_NONE	No rule
KLGPF_TABLE_RULE_SINGLE	———————————
KLGPF_TABLE_RULE_DOUBLE	═══════════════
KLGPF_TABLE_RULE_BROKEN	- -

Spacing **double** **5.0 Points**

If RuleType is KLGPF_TABLE_RULE_DOUBLE, Spacing is the amount of white space between the double rules. If RuleType is KLGPF_TABLE_RULE_BROKEN, Spacing is the dash length and the amount of white space between the rule's dashes.

Width **double** **0.0**

The Border's line width in document units. If *RuleType* is KLGPF_TABLE_RULE_DOUBLE, each line is equal to *Width*. **Note:** a setting of 0.0 and a *RuleType* other than KLGPF_TABLE_RULE_NONE produces the narrowest border that can be rendered.

Cell

A Cell object contains a non-flow Frame object into which table text is rendered. Cell attributes control the size of the cell, margins, borders, and spans.

Alignment **KlgPfCellAlignment** **KLGPF_CELL_ALIGNMENT_NONE**

Indicates the cell text's vertical alignment. If KLGPF_CELL_ALIGNMENT_NONE, the attribute setting is inherited from its parent object.

KLGPF_CELL_ALIGNMENT_NONE	Text is rendered at the top of the cell.
KLGPF_CELL_ALIGNMENT_TOP	Text is rendered at the top of the cell.
KLGPF_CELL_ALIGNMENT_CENTER	Text is rendered in the middle of the cell.

KLGPF_CELL_ALIGNMENT_BOTTOM	Text is rendered at the bottom of the cell.

BackgroundColor KlgPfObject NULL

Color object applied to the cell's background. If NULL, the attribute setting is inherited from its parent object.

BorderBottom KlgPfObject NULL

Border object applied to the cell's bottom border. If NULL, the attribute setting is inherited from its parent object.

BorderRight KlgPfObject NULL

Border object applied to the cell's right border. If NULL, the attribute setting is inherited from its parent object.

CellStyle KlgPfObject NULL

CellStyle provides the Style object applied to the rendered text in the cell. If NULL, the attribute setting is inherited from its parent object.

Frame KlgPfObject *Dynamic*

The cell's Frame object into which text is rendered. **Note:** you only get this attribute. You cannot set it.

Height double *Dynamic*

The cell's height in document units. By default the cell is sized to fit rendered text.

MarginBottom double 1 point

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's bottom edge.

MarginLeft double 1 point

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's left edge.

MarginRight double 1 point

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's right edge.

MarginTop double 1 point

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's top edge.

Spanned **Boolean** **FALSE**

If `TRUE`, indicates if the cell is part of a cell span.

SpanOrigin **KlgPfPoint** **(-1, -1)**

The row and column indices of the root cell of the span range.

SpanRows **int** **0**

The number of rows down the originator cell spans.

SpanColumns **int** **0**

The number of columns to the right the originator cell spans.

Column

A Column object defines borders, width, color, and alignment for all cells in that column. Unless explicitly set, the Column object inherits its attributes from the Table object. Where there is a conflict between Column and Row object attributes, priority is determined by the Table object's `Direction` attribute. In all cases, Cell object attributes take priority over Table, Column, and Row attributes.

Alignment **KlgPfCellAlignment** **KLGPF_CELL_ALIGNMENT_NONE**

Indicates the vertical alignment of text in each cell in the column. Unless explicitly set, `Alignment` is inherited from the Table object's `CellAlignment` attribute.

`KLGPF_CELL_ALIGNMENT_NONE`	Alignment is inherited.
`KLGPF_CELL_ALIGNMENT_TOP`	Text is rendered at the top of the cell.
`KLGPF_CELL_ALIGNMENT_CENTER`	Text is rendered in the middle of the cell.
`KLGPF_CELL_ALIGNMENT_BOTTOM`	Text is rendered at the bottom of the cell.

BackgroundColor **KlgPfObject** **NULL**

Color object applied to the column's entire background. If `NULL`, the attribute setting is inherited from its parent Table object.

BorderRight **KlgPfObject** **NULL**

Border object applied to borders between columns. If `NULL`, the attribute setting is inherited from its parent Table object.

CellMarginBottom **double** **NULL**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's bottom edge. If `NULL`, the attribute setting is inherited from its parent Table object.

CellMarginLeft **double** **NULL**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's left edge. If `NULL`, the attribute setting is inherited from its parent Table object.

CellMarginRight **double** **NULL**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's right edge. If `NULL`, the attribute setting is inherited from its parent Table object.

CellMarginTop **double** **NULL**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's top edge. If `NULL`, the attribute setting is inherited from its parent Table object.

CellStyle **KlgPfObject** **NULL**

`CellStyle` provides the Style object applied to the rendered cell text in the column. If `NULL`, the attribute setting is inherited from its parent Table object.

Width **double**

The column's width. Width is in document units.

Row

A Row object defines borders, height, color, and alignment for all cells in that row. Unless explicitly set, the Row object inherits its attributes from the Table object. Where there is a conflict between Row and Column object attributes, priority is determined by the Table object's `Direction` attribute. In all cases, Cell object attributes take priority over Table, Row, and Column attributes.

Alignment **KlgPfCellAlignment** **KLGPF_CELL_ALIGNMENT_NONE**

Indicates the vertical alignment of text in each cell in the row.

KLGPF_CELL_ALIGNMENT_NONE	Alignment is inherited.
KLGPF_CELL_ALIGNMENT_TOP	Text is rendered at the top of the cell.
KLGPF_CELL_ALIGNMENT_CENTER	Text is rendered in the middle of the cell.
KLGPF_CELL_ALIGNMENT_BOTTOM	Text is rendered at the bottom of the cell.

BackgroundColor **KlgPfObject** **NULL**

Color object applied to the row's entire background. If NULL, the attribute setting is inherited from its parent Table object.

BorderBottom **KlgPfObject** **NULL**

Border object applied to borders between rows.

CellList **KlgPfObject** **NULL**

A list of Cell objects in the row. **Note:** You can only get this attribute. You cannot set it directly.

CellMarginBottom **double** **NULL**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's bottom edge. If NULL, the attribute setting is inherited from its parent Table object.

CellMarginLeft **double** **NULL**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's left edge. If NULL, the attribute setting is inherited from its parent Table object.

CellMarginRight **double** **NULL**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's right edge. If NULL, the attribute setting is inherited from its parent Table object.

CellMarginTop **double** **NULL**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's top edge. If `NULL`, the attribute setting is inherited from its parent Table object.

CellStyle **KlgPfObject** **NULL**

`CellStyle` provides the Style object applied to the rendered cell text in the row. If `NULL`, the attribute setting is inherited from its parent Table object.

Table

A Table object defines many of the default settings for the table: borders, margins, color, and alignment. Row and Column objects inherit attribute settings from the Table object. When explicitly set, Row and Column object attributes take priority over the Table attributes.

Alignment **KlgPfParagraphAlignment** **KLGPF_ALIGN_CENTER**

The alignment of the entire table in the frame. Valid alignments are:

`KLGPF_ALIGN_LEFT`	The table is left aligned.
`KLGPF_ALIGN_CENTER`	The table is centered.
`KLGPF_ALIGN_RIGHT`	The table is right aligned. You cannot use tabs in a right-aligned paragraph.
`KLGPF_JUSTIFY`	The table is centered.

BackgroundColor **KlgPfObject** **NULL**

Color object applied to the table background.

BorderBottom **KlgPfObject** **Single border, Width 0.0, Black**

Border object applied to the table's bottom external border. By default, tables will have single, thin black borders.

BorderColumn **KlgPfObject** **Single border, Width 0.0, Black**

Border object applied to the table's internal vertical borders. By default, tables will have single, thin black borders.

BorderHeader **KlgPfObject** **Single border, Width 0.0, Black**

Border object applied to the header table's external border. By default, tables will have single, thin black borders.

BorderLeft **KlgPfObject** **Single border, Width 0.0, Black**

Border object applied to the table's left-side external border. By default, tables will have single, thin black borders.

BorderMode **KlgPfTableBorderMode** **KLGPF_TABLE_MODE_EXTERNAL_BORDERS**

If a table breaks across a page, *BorderMode* determines how the bottom and top borders between the break should be rendered. If the *OverflowMode* is set to KLGPF_TABLE_OVERFLOW_WRAP_COLUMNS, *BorderMode* also determines if the column borders should use the table's external borders or the inter-column borders in the event a column wraps.

KLGPF_TABLE_MODE_INTERNAL_BORDERS	If breaking across a page, indicates the bottom and top borders at the break should be rendered using the Row object's *BorderBottom* border attribute. If wrapping columns, indicates the column borders should be rendered using the *BorderColumn* attribute.
KLGPF_TABLE_MODE_EXTERNAL_BORDERS	If breaking across a page, indicates the bottom and top borders at the break should be rendered using the Table object's *BorderBottom* and *BorderTop* border attributes. If wrapping columns, indicates the column borders should be rendered using the *BorderRight* and *BorderLeft* attributes.

BorderRight **KlgPfObject** **Single border, Width 0.0, Black**

Border object applied to the table's right-side external border. By default, tables will have single, thin black borders.

BorderRow **KlgPfObject** **Single border, Width 0.0, Black**

Border object applied to the table's internal horizontal borders. By default, tables will have single, thin black borders.

BorderTop **KlgPfObject** **Single border, Width 0.0, Black**

Border object applied to the table's top external border. By default, tables will have single, thin black borders.

CellAlignment **KlgPfCellAlignment** **KLGPF_CELL_ALIGNMENT_NONE**

Indicates the vertical alignment of text in each cell in the table.

KLGPF_CELL_ALIGNMENT_NONE	Text is rendered at the top of the cell.
KLGPF_CELL_ALIGNMENT_TOP	Text is rendered at the top of the cell.
KLGPF_CELL_ALIGNMENT_CENTER	Text is rendered in the middle of the cell.
KLGPF_CELL_ALIGNMENT_BOTTOM	Text is rendered at the bottom of the cell.

ColumnList **KlgPfObject** ***dynamic***

Ordered list of all columns in the table. **Note:** You can only get this attribute. You cannot set it directly. It is created when the table is created.

CellMarginBottom **double** **1 point**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's bottom edge.

CellMarginLeft **double** **1 point**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's left edge.

CellMarginRight **double** **1 point**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's right edge.

CellMarginTop **double** **1 point**

Specifies the amount of internal white space, in document units, to leave between the cell's internal frame and the cell's top edge.

CellStyle **KlgPfObject** **Current Document Style**

CellStyle provides the Style object applied to the rendered cell text. *CellStyle* defaults to the Document object's *CurrentStyle* attribute.

DirectionPriority KlgPfTableDirectionPriority KLGPF_TABLE_PRIORITY_ROW

Indicates whether Column objects or Row objects should have priority when intersecting backgrounds and borders are drawn.

KLGPF_TABLE_PRIORITY_ROW	PageFormatter should use Row object attributes.
KLGPF_TABLE_PRIORITY_COLUMN	PageFormatter should use Column object attributes.

HeaderTable KlgPfObject Dynamic

A Table object used as the header.

OverflowMode KlgPfTableOverflowMode KLGPF_TABLE_OVERFLOW_CLIP_COLUMNS

Indicates what PageFormatter should do if a column extends over the page's right margin.

KLGPF_TABLE_OVERFLOW_CLIP_COLUMNS	Print as much of the table that fits across the page and clip the rest.
KLGPF_TABLE_OVERFLOW_WRAP_COLUMNS	PageFormatter should render the overflow columns on the next line.

RowList KlgPfObject *dynamic*

Ordered list of all rows in the table. **Note:** You can only get this attribute. You cannot set it directly.

P.3 Methods

KlgPfObjectIsTable()

Returns `True` if the object is a Table object.

```
Boolean
KlgPfObjectIsTable(
   KlgPfObject              object
   )
```

KlgPfObjectIsTableBorder()

Returns `True` if the object is a Border object.

```
Boolean
KlgPfObjectIsTableBorder(
   KlgPfObject              object
   )
```

KlgPfObjectIsTableCell()

Returns `True` if the object is a Cell object.

```
Boolean
KlgPfObjectIsTableCell(
   KlgPfObject              object
   )
```

KlgPfObjectIsTableColumn()

Returns `True` if the object is a Column object.

```
Boolean
KlgPfObjectIsTableColumn(
   KlgPfObject              object
   )
```

KlgPfObjectIsTableRow()

Returns `True` if the object is a Row object.

```
Boolean
KlgPfObjectIsTableRow(
   KlgPfObject              object
   )
```

KlgPfTableAddHeaderRow()

Adds a header row to a table. Returns `True` if successful.

```
Boolean
KlgPfTableAddHeaderRow(
   KlgPfObject              table,
   String                   *cell_data
   )
```

Parameters

table: The Table object.

cell_data: A string array. *cell_data* must contain as many entries as there are columns in the table. `NULL` represents a blank entry.

KlgPfTableAddRange()

Adds a series of cells to the specified row, beginning with the specified column. Returns `True` if successful.

```
Boolean
KlgPfTableAddRange(
   KlgPfObject              table,
   int                      row,
   int                      column,
   String                   *cell_data,
   int                      cell_count
   )
```

Parameters

table: The Table object.

row: The row to add a cell range to.

column: The first column in a row where the cell range is being added.

cell_data: A string array. `NULL` represents a blank entry.

cell_count: The number of cells to add.

KlgPfTableAddRow()

Adds a row to the end of the table. Returns `True` if successful.

```
Boolean
KlgPfTableAddRow(
   KlgPfObject              table,
   String                   *cell_data
   )
```

Parameters

table: The Table object.

cell_data: A string array. *cell_data* must contain as many entries as there are columns in the table. `NULL` represents a blank entry.

KlgPfTableBorderCopy()

```
KlgPfObject
KlgPfTableBorderCopy(
   KlgPfObject              source,
   KlgPfObject              new_parent
   )
```

Parameters

source: The Border object you wish to copy.

new_parent: The new Border object's parent object (can be a Table, Row, Column, or Cell object).

KlgPfTableBorderCreate()

Creates a Border object. Parent is the Table object.

```
KlgPfObject
KlgPfTableBorderCreate(
   KlgPfObject              parent
   )
```

KlgPfTableBorderDestroy()

Destroys a Border object. Note: you must destroy every border object you create.

```
void
KlgPfTableBorderDestroy(
   KlgPfObject              border_object
   )
```

KlgPfTableBorderGetColor()

Returns the Border object's Color object.

```
KlgPfObject
KlgPfTableBorderGetColor(
   KlgPfObject              border
   )
```

KlgPfTableBorderGetRuleType()

Retrieves the Border object's `RuleType` attribute.

```
KlgPfTableRuleType
KlgPfTableBorderGetRuleType(
   KlgPfObject              border
   )
```

KlgPfTableBorderGetSpacing()

Retrieves the Border object's `Spacing` attribute.

```
double
KlgPfTableBorderGetSpacing(
   KlgPfObject              border
   )
```

KlgPfTableBorderGetThickness()

Returns drawn width of a border, based on `Width`, `RuleType`, and `Spacing`. You should use *KlgPfTableBorderGetThickness()* to get the exact width of a border set as a double rule.

```
double
KlgPfTableBorderGetThickness(
   KlgPfObject              border
   )
```

KlgPfTableBorderGetWidth()

Returns the Border object's `Width` attribute.

```
double
KlgPfTableBorderGetWidth(
   KlgPfObject              border
   )
```

KlgPfTableBorderSetColor()

Sets the Border object's Color object. Returns `True` if successful.

```
Boolean
KlgPfTableBorderSetColor(
   KlgPfObject              border,
   KlgPfObject              color
   )
```

Parameters

border: A Border object.

color: A Color object.

KlgPfTableBorderSetRuleType()

Sets the Border object's `RuleType` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableBorderSetRuleType(
   KlgPfObject              border,
   KlgPfTableRuleType       type
   )
```

Parameters

border: A Border object.

type: Ruling type the border should have (for example, `KLGPF_TABLE_RULE_SINGLE`).

KlgPfTableBorderSetSpacing()

Sets the Border object's `Spacing` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableBorderSetSpacing(
   KlgPfObject              border,
   double                   spacing
   )
```

Parameters

border: A Border object.

spacing: If `RuleType` is `KLGPF_TABLE_RULE_DOUBLE`, `spacing` is the amount of white space between the double rules. If `RuleType` is `KLGPF_TABLE_RULE_BROKEN`, `Spacing` is the length of the rule's dashes and the amount of white space between the dashes.

KlgPfTableBorderSetWidth()

Sets the Border object's `Width` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableBorderSetWidth(
   KlgPfObject              border,
   double                   width
   )
```

Parameters

border: A Border object.

width: The border's width, in document units.

KlgPfTableCellGetAlignment()

Retrieves the Cell object's `Alignment` attribute.

```
KlgPfCellAlignment
KlgPfTableCellGetAlignment(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetBackgroundColor()

Retrieves the Cell object's `BackgroundColor` attribute. *KlgPfTableCellGetBackgroundColor()* returns the Color object used for the cell's background.

```
KlgPfObject
KlgPfTableCellGetBackgroundColor(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetBorder()

Retrieves the Cell object's bottom or right Border object. If no Border object is specified for the given border location, *KlgPfTableCellGetBorder()* will return the inherited border object.

```
KlgPfObject
KlgPfTableCellGetBorder(
   KlgPfObject              cell,
   KlgPfTableBorderLocation  border_location
   )
```

Parameters

cell: The Cell object.

border_location: The location of the desired border object. *border* can be:

`KLGPF_TABLE_BORDER_BOTTOM`

`KLGPF_TABLE_BORDER_RIGHT`

KlgPfTableCellGetFrame()

Retrieves the Cell object's *Frame* attribute. *KlgPfTableCellGetFrame()* returns a Frame object.

```
KlgPfObject
KlgPfTableCellGetFrame(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetHeight()

Retrieves the Cell object's `Height` attribute.

```
double
KlgPfTableCellGetHeight(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetMarginBottom()

Returns the Cell object's bottom margin, in document units.

```
double
KlgPfTableCellGetMarginBottom(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetMarginLeft()

Returns the Cell object's left margin, in document units.

```
double
KlgPfTableCellGetMarginLeft(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetMarginRight()

Returns the Cell object's right margin, in document units.

```
double
KlgPfTableCellGetMarginRight(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetMarginTop()

Returns the Cell object's top margin, in document units.

```
double
KlgPfTableCellGetMarginTop(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetSpanColumns()

Returns the number of columns to the right the cell spans. Returns -1 on error.

```
int
KlgPfTableCellGetSpanColumns(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetSpanOrigin()

Returns the row and column indices of the cell originating the span. If *KlgPfTableCellGetSpanOrigin()* returns `(-1, -1)`, the cell is not a span originator.

```
KlgPfPoint
KlgPfTableCellGetSpanOrigin(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetSpanRows()

Returns the number of rows down the cell spans. Returns -1 on error.

```
int
KlgPfTableCellGetSpanRows(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetStyle()

Retrieves the Cell object's `CellStyle` attribute. If no Style object has been specified for *Cellstyle*, *KlgPfTableCellGetStyle()* will attempt to return the Style object for the Cell's parent objects in the following order: Row object, Column object, and finally Table object.

```
KlgPfObject
KlgPfTableCellGetStyle(
   KlgPfObject              cell
   )
```

KlgPfTableCellGetWidth()

Retrieves the Cell object's *Width* attribute.

```
double
KlgPfTableCellGetWidth(
   KlgPfObject              cell
   )
```

KlgPfTableCellIsSpanned()

Returns `True` if the cell is the originator of a cell span.

```
Boolean
KlgPfTableCellIsSpanned(
   KlgPfObject              cell
   )
```

KlgPfTableCellSetAlignment()

Sets the Cell object's `Alignment` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableCellSetAlignment(
   KlgPfObject              cell,
   KlgPfCellAlignment       alignment
   )
```

Parameters

cell: The Cell object.

alignment: A cell alignment constant (for example, `KLGPF_CELL_ALIGNMENT_CENTER`).

KlgPfTableCellSetBackgroundColor()

Sets the Cell object's `BackgroundColor` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableCellSetBackgroundColor(
   KlgPfObject              cell,
   KlgPfObject              color
   )
```

Parameters

cell: The Cell object.

color: A Color object.

KlgPfTableCellSetBorder()

Sets the Cell object's border style. Returns `True` if successful.

```
Boolean
KlgPfTableCellSetBorder(
   KlgPfObject               cell,
   KlgPfTableBorderLocation  border_location,
   KlgPfObject               border_style
   )
```

Parameters

cell: The Cell object.

border_location: The side of the cell on which to use the given border object. *border* can be:

`KLGPF_TABLE_BORDER_BOTTOM`

`KLGPF_TABLE_BORDER_RIGHT`

border_style: A Border object

KlgPfTableCellSetHeight()

Sets the Cell object's `Height` attribute. **Note:** by default cell height is dynamic, sized by PageFormatter for the rendered text. If you explicitly set a cell height, the cell will no longer dynamically size. Returns `True` if successful.

```
Boolean
KlgPfTableCellSetHeight(
   KlgPfObject               cell,
   double                    height
   )
```

Parameters

cell: The Cell object.

height: The cell's height in document units.

KlgPfTableCellSetMarginBottom()

Sets the Cell object's `MarginBottom` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableCellSetMarginBottom(
   KlgPfObject               cell,
   double                    margin_width
   )
```

Parameters

cell: The Cell object.

margin_width: The cell's internal margin in document units.

KlgPfTableCellSetMarginLeft()

Sets the Cell object's `MarginLeft` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableCellSetMarginLeft(
   KlgPfObject               cell,
   double                    margin_width
```

```
   )
```

Parameters

cell: The Cell object.

margin_width: The cell's internal margin in document units.

KlgPfTableCellSetMarginRight()

Sets the Cell object's `MarginRight` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableCellSetMarginRight(
   KlgPfObject              cell,
   double                   margin_width
   )
```

Parameters

cell: The Cell object.

margin_width: The cell's internal margin in document units.

KlgPfTableCellSetMarginTop()

Sets the Cell object's `MarginTop` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableCellSetMarginTop(
   KlgPfObject              cell,
   double                   margin_width
   )
```

Parameters

cell: The Cell object.

margin_width: The cell's internal margin in document units.

KlgPfTableCellSetStyle()

Sets the Cell object's `Style` attribute. The Style object will be applied to text rendered when printing to a cell. Returns `True` if successful.

```
Boolean
KlgPfTableCellSetStyle(
   KlgPfObject              cell,
   KlgPfObject              style
   )
```

Parameters

cell: The Cell object.

style: A Style object.

KlgPfTableColumnGetAlignment()

Retrieves the Column object's `Alignment` attribute.

```
KlgPfCellAlignment
KlgPfTableColumnGetAlignment(
   KlgPfObject              column
   )
```

KlgPfTableColumnGetBackgroundColor()

Retrieves the Column object's `BackgroundColor` attribute. *KlgPfTableColumnGetBackgroundColor()* returns the Color object used for the column's background.

```
KlgPfObject
KlgPfTableColumnGetBackgroundColor(
   KlgPfObject              column
   )
```

KlgPfTableColumnGetBorder()

Retrieves the Column object's `BorderRight` attribute. *KlgPfTableColumnGetBorder()* returns the Border object applied to the column's right border.

```
KlgPfObject
KlgPfTableColumnGetBorder(
   KlgPfObject              column
   )
```

KlgPfTableColumnGetCellMarginBottom()

Retrieves the Column object's `CellMarginBottom` attribute.

```
double
KlgPfTableColumnGetCellMarginBottom(
   KlgPfObject              column
   )
```

KlgPfTableColumnGetCellMarginLeft()

Retrieves the Column object's `CellMarginLeft` attribute.

```
double
KlgPfTableColumnGetCellMarginLeft(
   KlgPfObject              column
   )
```

KlgPfTableColumnGetCellMarginRight()

Retrieves the Column object's `CellMarginRight` attribute.

```
double
KlgPfTableColumnGetCellMarginRight(
   KlgPfObject              column
   )
```

KlgPfTableColumnGetCellMarginTop()

Retrieves the Column object's `CellMarginTop` attribute.

```
double
KlgPfTableColumnGetCellMarginTop(
   KlgPfObject              column
   )
```

KlgPfTableColumnGetCellStyle()

Retrieves the Column object's `CellStyle` attribute. Returns the Style object applied to text in the column cell rendered with the *KlgPfTablePrintToCell()* method.

```
KlgPfObject
KlgPfTableColumnGetCellStyle(
   KlgPfObject              column
   )
```

KlgPfTableColumnGetWidth()

Retrieves the Column object's `Width` attribute.

```
double
KlgPfTableColumnGetWidth(
   KlgPfObject              column
   )
```

KlgPfTableColumnSetAlignment()

Sets the Column object's `Alignment` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableColumnSetAlignment(
   KlgPfObject              column,
   KlgPfCellAlignment       alignment
   )
```

Parameters

column: A Column object.

alignment: A cell alignment constant (for example, `KLGPF_CELL_ALIGNMENT_CENTER`).

KlgPfTableColumnSetBackgroundColor()

Sets the Column object's `BackgroundColor` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableColumnSetBackgroundColor(
   KlgPfObject              column,
   KlgPfObject              color
   )
```

Parameters

column: A Column object.

color: A Color object.

KlgPfTableColumnSetBorder()

Sets the Column object's `BorderRight` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableColumnSetBorder(
   KlgPfObject              column,
   KlgPfObject              border_style
   )
```

Parameters

column: A Column object.

border_style: A Border object.

KlgPfTableColumnSetCellMarginBottom()

Sets the Column object's `CellMarginBottom` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableColumnSetCellMarginBottom(
   KlgPfObject              column,
   double                   margin_width
   )
```

Parameters

column: A Column object.

margin_width: The cell's internal margin, in document units.

KlgPfTableColumnSetCellMarginLeft()

Sets the Column object's `CellMarginLeft` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableColumnSetCellMarginLeft(
   KlgPfObject              column,
   double                   margin_width
   )
```

Parameters

column: A Column object.

margin_width: The cell's internal margin, in document units.

KlgPfTableColumnSetCellMarginRight()

Sets the Column object's `CellMarginRight` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableColumnSetCellMarginRight(
   KlgPfObject              column,
   double                   margin_width
   )
```

Parameters

column: A Column object.

margin_width: The cell's internal margin, in document units.

KlgPfTableColumnSetCellMarginTop()

Sets the Column object's `CellMarginTop` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableColumnSetCellMarginTop(
   KlgPfObject              column,
   double                   margin_width
   )
```

Parameters

column: A Column object.

margin_width: The cell's internal margin, in document units.

KlgPfTableColumnSetCellStyle()

Sets the Column object's `CellStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableColumnSetCellStyle(
   KlgPfObject              column,
   KlgPfObject              style
   )
```

Parameters

column: A Column object.

style: The Style object to be applied to all rendered text in the column when printing to a cell.

KlgPfTableCopy()

Returns a copy of the specified Table object, its child objects, and contents.

```
KlgPfObject
KlgPfTableCopy(
   KlgPfObject              source
   )
```

KlgPfTableCreate()

Creates and returns a Table object.

```
KlgPfObject
KlgPfTableCreate(
   KlgPfObject               parent,
   double                   *column_widths
   int                       ncolumns
)
```

Parameters

parent: The Frame or Document object to create the Table object in. If you pass a Document object, the table will be added to the flow in the current frame. The Table object is added to the Document object's *TableList* list.

column_widths: An array of column widths, terminated by `-1`. *KlgPfTableCreate()* creates as many columns in the table as there are widths in the array.

ncolumns: The number of columns to create.

KlgPfTableCreateSimple()

Creates a Table object and populates it with an array of string data. The table is printed and then destroyed. Returns `True` if successful.

```
Boolean
KlgPfTableCreateSimple(
   KlgPfObject                parent,
   double                    *column_widths,
   int                        ncolumns
   String                   **cell_data
   )
```

Parameters

parent: The Frame or Document object to create the Table object in. If you pass a Document object, the table will be added to the flow in the current frame. The Table object is added to the Document object's *TableList* list.

column_widths: An array of column widths. *KlgPfTableCreate()* creates as many columns in the table as there are widths in the array.

ncolumns: The number of columns to create.

cell_data: A 2D array of Strings to be used as cell contents. Each row of the array must have as many entries as there are columns in the table. `NULL` indicates a blank cell. You must terminate the array with a row of `NULL`s.

KlgPfTableDestroy()

Destroys the Table object and all of its children.

```
void
KlgPfTableDestroy(
   KlgPfObject              table
   )
```

KlgPfTableGetAlignment()

Retrieves the Table object's `Alignment` attribute, indicating the table's horizontal alignment in a frame.

```
KlgPfParagraphAlignment
KlgPfTableGetAlignment(
   KlgPfObject              table
   )
```

KlgPfTableGetBackgroundColor()

Retrieves the Table object's `BackgroundColor` attribute. *KlgPfTableGetBackgroundColor()* returns the Color object applied to the table's background.

```
KlgPfObject
KlgPfTableGetBackgroundColor(
   KlgPfObject              table
   )
```

KlgPfTableGetBorderMode()

Returns the Table object's `BorderMode` attribute. See the Table object's *BorderMode* attribute on page 266 for more information.

```
KlgPfTableBorderMode
KlgPfTableGetBorderMode(
   KlgPfObject              table
   )
```

KlgPfTableGetBorder()

Returns a specified Table object border attribute.

```
KlgPfObject
KlgPfTableGetBorder(
   KlgPfObject              table,
   KlgPfTableBorderLocation  border
   )
```

Parameters

table: The Table object.

border: The location of the desired border object. *border* can be:

`KLGPF_TABLE_BORDER_TOP`	Return the `BorderTop` attribute.
`KLGPF_TABLE_BORDER_BOTTOM`	Return the `BorderBottom` attribute.
`KLGPF_TABLE_BORDER_LEFT`	Return the `BorderLeft` attribute.
`KLGPF_TABLE_BORDER_RIGHT`	Return the `BorderRigbt` attribute.
`KLGPF_TABLE_BORDER_ROW`	Return the `BorderRow` attribute.
`KLGPF_TABLE_BORDER_COLUMN`	Return the `BorderColumn` attribute.
`KLGPF_TABLE_BORDER_HEADER`	Return the `BorderHeader` attribute.

KlgPfTableGetCell()

Retrieves a specified Cell object. For example, *KlgPfTableGetCell(table, 0, 0)* would return the upper left cell in the table. If the cell or row does not exist, *KlgPfTableGetCell()* will create the cell or row and return the cell. If the requested cell is obscured by a span, *KlgPfTableGetCell()* returns the span's originating cell.

```
KlgPfObject
KlgPfTableGetCell(
   KlgPfObject              table,
   int                      row,
   int                      column
   )
```

Parameters

table: The Table object.

row: The row index of the desired cell.

column: The column index of the desired cell.

KlgPfTableGetCellAlignment()

Retrieves the Table object's `CellAlignment` attribute.

```
KlgPfCellAlignment
KlgPfTableGetCellAlignment(
   KlgPfObject              table
   )
```

KlgPfTableGetCellFrame()

Retrieves a specified Cell object's Frame object. For example, *KlgPfTableGetCellFrame(table, 0, 0)* would return the Frame object of the first cell in the table. If the cell does not exist, *KlgPfTableGetCellFrame()* will create the row and return the cell. If the requested cell is obscured by a span, *KlgPfTableGetCellFrame()* returns the span's originating cell.

```
KlgPfObject
KlgPfTableGetCellFrame(
   KlgPfObject              table,
   int                      row,
   int                      column
   )
```

Parameters

table: The Table object.

row: The row index of the desired cell.

column: The column index of the desired cell.

KlgPfTableGetCells()

Retrieves a list of cells in the indexed row of the table. For example, *KlgPfTableGetCells(table, 0)* would return a List object containing all the Cell objects in the first row. **Note**: The returned List object may contain cells which are obscured by a span.

```
KlgPfObject
KlgPfTableGetCells(
   KlgPfObject              table,
   int                      row
   )
```

Parameters

table: The Table object.

row: The index of the desired row.

KlgPfTableGetColumn()

Retrieves a Column object.

```
KlgPfObject
KlgPfTableGetColumn(
   KlgPfObject              table,
   int                      column
   )
```

Parameters

table: The Table object.

column: The index of the desired column.

KlgPfTableGetColumnList()

Retrieves the Table object's `ColumnList` attribute, an ordered list containing all the Table's Column objects. A List object is returned.

```
KlgPfObject
KlgPfTableGetColumnList(
   KlgPfObject              table
   )
```

KlgPfTableGetCellMarginBottom()

Retrieves the Table's `CellMarginBottom` attribute.

```
double
KlgPfTableGetCellMarginBottom(
   KlgPfObject              table
   )
```

KlgPfTableGetCellMarginLeft()

Retrieves the Table's `CellMarginLeft` attribute.

```
double
KlgPfTableGetCellMarginLeft(
   KlgPfObject              table
   )
```

KlgPfTableGetCellMarginRight()

Retrieves the Table's `CellMarginRight` attribute.

```
double
KlgPfTableGetCellMarginRight(
   KlgPfObject              table
   )
```

KlgPfTableGetCellMarginTop()

Retrieves the Table's `CellMarginTop` attribute.

```
double
KlgPfTableGetCellMarginTop(
   KlgPfObject              table
   )
```

KlgPfTableGetCellStyle()

Retrieves the Table object's `CellStyle` attribute. Returns the Style object applied to text in table cells rendered with the *KlgPfTablePrintToCell()* method.

```
KlgPfObject
KlgPfTableGetCellStyle(
   KlgPfObject              table
   )
```

KlgPfTableGetHeaderTable()

Retrieves the Table object's `HeaderTable` attribute. *KlgPfTableGetHeaderTable()* returns a Table object. If no header table has been created for the Table object, *KlgPfTableGetHeaderTable()* will create and return an empty header table.

```
KlgPfObject
KlgPfTableGetHeaderTable(
   KlgPfObject              table
   )
```

KlgPfTableGetOverflowMode()

Retrieves the Table object's `OverflowMode` attribute. For more information on Overflow Mode, see the Table object's `OverflowMode` attribute on page 268.

```
KlgPfTableOverflowMode
KlgPfTableGetOverflowMode(
   KlgPfObject              table
   )
```

KlgPfTableGetPriorityDirection()

Retrieves the Table object's `Direction` attribute. For more information on direction priority, see the Table object's `Direction` attribute on page 268.

```
KlgPfTableDirectionPriority
KlgPfTableGetPriorityDirection(
   KlgPfObject              table
   )
```

KlgPfTableGetRow()

Retrieves a specified Row object from the `RowList` attribute. For example, *KlgPfTableGetRow(table, 0)* would return the first row.

```
KlgPfObject
KlgPfTableGetRow(
   KlgPfObject              table,
   int                      row
   )
```

Parameters

table: The Table object.

row: The index of the desired row. If `row` is greater than the current number of rows, *KlgPfTableGetRow()* will create and return the requested row, adding as many blank rows as necessary after the last row in the table.

KlgPfTableGetRowList()

Retrieves the Table object's `RowList` attribute, an ordered list containing all the Table's Row objects. A List object is returned.

```
KlgPfObject
KlgPfTableGetRowList(
   KlgPfObject              table
   )
```

KlgPfTablePrintAndDestroy()

Renders the Table object in the specified frame and removes it and its child objects from memory. Returns `True` if successful.

```
Boolean
KlgPfTablePrintAndDestroy(
   KlgPfObject              parent,
   KlgPfObject              table
   )
```

Parameters

parent: The Document or Frame object to render the table in.

table: The Table object.

KlgPfTablePrintToCell()

Prints to a specific cell. For example, *KlgPfTablePrintToCell(table, 0, 0, "Hello, printer!")* would render "Hello, printer!" in the upper left cell. Returns `True` if successful.

```
Boolean
KlgPfTablePrintToCell(
   KlgPfObject              table,
   int                      row,
   int                      column,
   String                   cell_data
   )
```

Parameters

table: The Table object.

row: The row index of the desired cell.

column: The column index of the desired cell.

cell_data: String you wish to render.

KlgPfTableRowGetAlignment()

Retrieves the Row object's `Alignment` attribute. If no row alignment has been set, *KlgPfTableRowGetAlignment()* returns the Table object's `Alignment` attribute.

```
KlgPfCellAlignment
KlgPfTableRowGetAlignment(
   KlgPfObject              row
   )
```

KlgPfTableRowGetBackgroundColor()

Retrieves the Color object set for the Row object's `BackgroundColor` attribute. If no Color object has been set, *KlgPfTableRowGetBackgroundColor()* returns the Table object's `BackgroundColor` attribute.

```
KlgPfObject
KlgPfTableRowGetBackgroundColor(
   KlgPfObject              row
   )
```

KlgPfTableRowGetBorder()

Retrieves the Border object set for the Row object's `BottomBorder` attribute. If no Border object has been set, *KlgPfTableRowGetBorder()* returns the Table object's *BorderRow* attribute.

```
KlgPfObject
KlgPfTableRowGetBorder(
   KlgPfObject              row
   )
```

KlgPfTableRowGetCellList()

Retrieves the Row object's `CellList` attribute. *KlgPfTableRowGetCellList()* returns a List object. **Note**: The List object may contain cells which are obscured by a span.

```
KlgPfObject
KlgPfTableRowGetCellList(
   KlgPfObject              row
   )
```

KlgPfTableRowGetCellMarginBottom()

Retrieves the Row object's `CellMarginBottom` attribute.

```
double
KlgPfTableRowGetCellMarginBottom(
   KlgPfObject              row
   )
```

KlgPfTableRowGetCellMarginLeft()

Retrieves the Row object's `CellMarginLeft` attribute.

```
double
KlgPfTableRowGetCellMarginLeft(
   KlgPfObject              row
   )
```

KlgPfTableRowGetCellMarginRight()

Retrieves the Row object's `CellMarginRight` attribute.

```
double
KlgPfTableRowGetCellMarginRight(
   KlgPfObject              row
   )
```

KlgPfTableRowGetCellMarginTop()

Retrieves the Row object's `CellMarginTop` attribute.

```
double
KlgPfTableRowGetCellMarginTop(
   KlgPfObject              row
   )
```

KlgPfTableRowGetCellStyle()

Retrieves the Row object's `CellStyle` attribute.

```
KlgPfObject
KlgPfTableRowGetCellStyle(
   KlgPfObject               row
   )
```

KlgPfTableRowSetAlignment()

Sets the Row object's `Alignment` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableRowSetAlignment(
   KlgPfObject               row,
   KlgPfCellAlignment        alignment
   )
```

Parameters

row: A Row object.

alignment: A cell alignment constant (for example, `KLGPF_CELL_ALIGNMENT_CENTER`).

KlgPfTableRowSetBackgroundColor()

Sets the Row object's `BackgroundColor` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableRowSetBackgroundColor(
   KlgPfObject               row,
   KlgPfObject               color
   )
```

Parameters

row: A Row object.

color: A Color object.

KlgPfTableRowSetBorder()

Sets the Row object's `BorderBottom` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableRowSetBorder(
   KlgPfObject               row,
   KlgPfObject               border_style
   )
```

Parameters

row: A Row object.

border_style: A Border object.

KlgPfTableRowSetCellMarginBottom()

Sets the Row object's `CellMarginBottom` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableRowSetCellMarginBottom(
   KlgPfObject               row,
   double                    margin_width
   )
```

Parameters

row: The Row object.

margin_width: The cell's internal margin in document units.

KlgPfTableRowSetCellMarginLeft()

Sets the Row object's `CellMarginLeft` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableRowSetCellMarginLeft(
   KlgPfObject              row,
   double                   margin_width
   )
```

Parameters

row: The Row object.

margin_width: The cell's internal margin in document units.

KlgPfTableRowSetCellMarginRight()

Sets the Row object's `CellMarginRight` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableRowSetCellMarginRight(
   KlgPfObject              row,
   double                   margin_width
   )
```

Parameters

row: The Row object.

margin_width: The cell's internal margin in document units.

KlgPfTableRowSetCellMarginTop()

Sets the Row object's `CellMarginTop` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableRowSetCellMarginTop(
   KlgPfObject              row,
   double                   margin_width
   )
```

Parameters

row: The Row object.

margin_width: The cell's internal margin in document units.

KlgPfTableRowSetCellStyle()

Sets the Row object's `CellStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableRowSetCellStyle(
   KlgPfObject              row,
   KlgPfObject              style
   )
```

Parameters

row: A Row object.

style: The Style object to be applied to all rendered text in the row when printing to a cell.

KlgPfTableSetAlignment()

Sets the Table object's `Alignment` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableSetAlignment(
   KlgPfObject              table,
   KlgPfParagraphAlignment  alignment
   )
```

Parameters

table: A Table object.

alignment: The Table's horizontal alignment in a frame (for example, `KLGPF_CELL_ALIGNMENT_CENTER`).

KlgPfTableSetBackgroundColor()

Sets the Table object's `BackgroundColor` attribute using the passed Color object. Returns `True` if successful.

```
Boolean
KlgPfTableSetBackgroundColor(
   KlgPfObject              table,
   KlgPfObject              color
   )
```

Parameters

table: A Table object.

color: A Color object.

KlgPfTableSetBorderMode()

Sets if the borders between the break should be rendered using the table's external borders or the inter-row/column border. Returns `True` if successful.

```
Boolean
KlgPfTableSetBorderMode(
   KlgPfObject              table,
   KlgPfTableBorderMode     border_mode
   )
```

Parameters

table: The Table object.

border_mode: Borders to render between breaks. *border_mode* can be:

KLGPF_TABLE_MODE_INTERNAL_BORDERS	If breaking across a page, indicates the bottom and top borders at the break should be rendered using the Row object's *BorderRow* border attribute. If wrapping columns, indicates the column borders should be rendered using the *BorderColumn* attribute.
KLGPF_TABLE_MODE_EXTERNAL_BORDERS	If breaking across a page, indicates the bottom and top borders at the break should be rendered using the Table object's *BorderBottom* and *BorderTop* border attributes. If wrapping columns, indicates the column borders should be rendered using the *BorderRight* and *BorderLeft* attributes.

KlgPfTableSetBorder()

Sets the specified border attribute. Returns `True` if successful.

```
Boolean
KlgPfTableSetBorder(
   KlgPfObject               table,
   KlgPfTableBorderLocation  border_location,
   KlgPfObject               border_style
   )
```

Parameters

table: The Table object.

border_location: The location of the desired border object. *border* can be:

KLGPF_TABLE_BORDER_TOP	Sets the *BorderTop* attribute.
KLGPF_TABLE_BORDER_BOTTOM	Sets the *BorderBottom* attribute.
KLGPF_TABLE_BORDER_LEFT	Sets the *BorderLeft* attribute.
KLGPF_TABLE_BORDER_RIGHT	Sets the *BorderRigbt* attribute.
KLGPF_TABLE_BORDER_ROW	Sets the *BorderRow* attribute.
KLGPF_TABLE_BORDER_COLUMN	Sets the *BorderColumn* attribute.
KLGPF_TABLE_BORDER_HEADER	Sets the *BorderHeader* attribute.

border_style: A Border object.

KlgPfTableSetCellAlignment()

Sets the Table object's `CellAlignment` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableSetCellAlignment(
   KlgPfObject              table,
   KlgPfCellAlignment       alignment
   )
```

Parameters

table: A Table object.

alignment: A cell alignment constant (for example, `KLGPF_CELL_ALIGNMENT_CENTER`).

KlgPfTableSetCellMarginBottom()

Sets the Table object's `CellMarginBottom` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableSetCellMarginBottom(
   KlgPfObject              table,
   double                   margin_width
   )
```

Parameters

table: The Table object.

margin_width: The cell's internal margin in document units.

KlgPfTableSetCellMarginLeft()

Sets the Table object's `CellMarginLeft` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableSetCellMarginLeft(
   KlgPfObject              table,
   double                   margin_width
   )
```

Parameters

table: The Table object.

margin_width: The cell's internal margin in document units.

KlgPfTableSetCellMarginRight()

Sets the Table object's `CellMarginRight` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableSetCellMarginRight(
   KlgPfObject              table,
   double                   margin_width
   )
```

Parameters

table: The Table object.

margin_width: The cell's internal margin in document units.

KlgPfTableSetCellMarginTop()

Sets the Table object's `CellMarginTop` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableSetCellMarginTop(
   KlgPfObject              table,
   double                   margin_width
   )
```

Parameters

table: The Table object.

margin_width: The cell's internal margin in document units.

KlgPfTableSetCellStyle()

Sets the Table object's `CellStyle` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableSetCellStyle(
   KlgPfObject              table,
   KlgPfObject              style
   )
```

Parameters

table: The Table object.

style: A Style object.

KlgPfTableSetOverflowMode()

Sets the Table object's `OverflowMode` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableSetOverflowMode(
   KlgPfObject              table,
   KlgPfTableOverflowMode   overflow_mode
   )
```

Parameters

table: The Table object.

overflow_mode: An Overflow Mode constant: `KLGPF_TABLE_MODE_INTERNAL_BORDERS` or `KLGPF_TABLE_MODE_EXTERNAL_BORDERS`.

KlgPfTableSetPriorityDirection()

Sets the Table object's `Direction` attribute. Returns `True` if successful.

```
Boolean
KlgPfTableSetPriorityDirection(
   KlgPfObject                  table,
   KlgPfTableDirectionPriority  direction
   )
```

Parameters

table: The Table object.

direction: A Direction Priority constant: `KLGPF_TABLE_PRIORITY_ROW` or `KLGPF_TABLE_PRIORITY_COLUMN`.

KlgPfTableSpanCells()

Sets a specified cell to span a range of other cells. For example, *KlgPfTableSpanCells(table, 0, 0, 2, 3)* would set the first cell in the table to span the first two rows and the first three columns. *KlgPfTableSpanCells()* returns a Cell object or `NULL` on failure. *KlgPfTableSpanCells()* will fail if a span extends beyond a valid column or it would overlap an existing span.

```
KlgPfObject
KlgPfTableSpanCells(
   KlgPfObject             table,
   int                     start_row,
   int                     start_column,
   int                     rows,
   int                     columns
   )
```

Parameters

table: The Table object.

start_row: The row index of the span-originating cell.

start_column: The column index of the span-originating cell.

rows: The number of rows down the cell should span.

columns: The number of columns to the right the cell should span.

Q

Functions and Macros

The following functions and macros are provided for convenience. They are defined in *KlgPf.h*.

KlgPfConvertCmToInches()

Returns the number of inches in the specified number of centimeters (1 inch is defined as equal to 2.54 centimeters).

```
double
KlgPfConvertCmToInches(
   double        cm
   )
```

KlgPfConvertCmToPoints()

Returns the number of points in the specified number of centimeters (1 centimeter is defined as *points per inch*/*centimeters per inch*).

```
double
KlgPfConvertCmToPoints(
   double        cm
   )
```

KlgPfConvertInchesToCm()

Returns the number of centimeters in the specified number of inches (1 inch is defined as equal to 2.54 centimeters).

```
double
KlgPfConvertInchesToCm(
   double        inches
   )
```

KlgPfConvertInchesToPoints()

Returns the number of points in the specified number of inches (1 inch is defined as equal to 72 points).

```
double
KlgPfConvertInchesToPoints(
   double        inches
   )
```

KlgPfConvertPointsToCm()

Returns the number of centimeters in the specified number of points (1 centimeter is defined as `POINTS PER INCH / CENTIMETERS PER INCH`).

```
double
KlgPfConvertPointsToCm(
   double        points
   )
```

KlgPfConvertPointsToInches()

Returns the number of inches in the specified number of points (1 inch is defined as equal to 72 points).

```
double
KlgPfConvertPointsToInches(
   double        points
   )
```

KlgPfError()

KlgPfError() prints a warning message to *stderr* indicating the errors location and an error message. This is the default error handler.

```
void
KlgPfError(
   KlgPfErrorStruct  *error,
   XtPointer         user_data
   )
```

Parameters

error: A handle to a `KlgPfErrorStruct` structure:

```
typedef struct {
   int           error_level;    /* read only */
   String        location;       /* read only */
   String        message;        /* read only */
   Boolean       exit;
} KlgPfErrorStruct;
```

user_data: A handle to user data. It is passed to the error handler. This default error handler ignores this parameter.

KlgPfErrorGetHandler()

Returns current error handling procedure.

```
KlgPfErrorProc
KlgPfErrorGetHandler()
```

KlgPfErrorGetUserData()

Returns any specified user data. By default user data is `NULL`.

```
XtPointer
KlgPfErrorGetUserData()
```

KlgPfErrorSetHandler()

Sets your own error handling procedure.

```
Boolean
KlgPfErrorSetHandler(
   KlgPfErrorProc    error_handler
   )
```

Parameters

error_handler: A procedure that conforms to the following type definition:

```
typedef void (*KlgPfErrorProc)(
   KlgPfErrorStruct  *error,
   XtPointer         user_data
   )
```

KlgPfErrorSetUserData()

Sets a handle to user defined data that will be passed to the current error handler method. By default this is *KlgPfError()* which ignores user data.

```
Boolean
KlgPfErrorSetUserData(
   XtPointer     user_data
   )
```

KlgPfFilterSimple()

KlgPfFilterSimple() accepts input from a string buffer, a text file or a stream and flows the text to the supplied document object.

```
Boolean
KlgPfFilterSimple(
   KlgPfObject   doc,
   KlgPfIOType   io_type,
   XtPointer     data,
   Boolean       escape_macro_symbols,
   Boolean       lines_are_paragraphs
   )
```

Returns

Returns `True` if the flow is rendered successfully.

Parameters

doc: The Document object.

io_type: Should be either `KlgPfIOStream`, `KlgPfIOFilename`, or `KlgPfIOBuffer`.

data: A handle to data you want to flow. This is a stream, a file, or a buffer depending on *io_type*.

escape_macro_symbols: If `True`, *KlgPfFilterSimple()* will not interpret @ symbols as macros.

lines_are_paragraphs: If `True`, *KlgPfFilterSimple()* calls a *KlgPfFlowLineFeed()* when reaching the end of a line. (This results in a new paragraph for each line.) If `False`, a new paragraph is generated only if there is a blank line in the input.

KlgPfGlobalGet()

Returns the specified global variable's current value. This value is either a directory or file name.

```
XtPointer
KlgPfGlobalGet(
   KlgPfGlobalType   global
   )
```

Parameters

global: The global variable you want to get. *global* can have one of the following values:

Value	Explanation
`KLGPF_GLOBAL_AFM_DIR`	Directory containing the Adobe AFM (font metric) files.
`KLGPF_GLOBAL_PPD_DIR`	Directory containing the Adobe PPD (printer descriptions) files.
`KLGPF_GLOBAL_PRINTERS_FILE`	File that associates printer with PPD files.
`KLGPF_GLOBAL_TFM_DIR`	Directory containing HP TFM (font metric) files.

KlgPfGlobalSet()

Sets the specified global variable's value. The value is either a directory or a file name.

```
Boolean
KlgPfGlobalSet(
   KlgPfGlobalType   global
   XtPointer         value
   )
```

Parameters

value: The directory or file location for the given global variable.

global: The global variable you want to set. *global* can have one of the following values:

Value	Explanation
`KLGPF_GLOBAL_AFM_DIR`	Directory containing the Adobe AFM (font metric) files.
`KLGPF_GLOBAL_PPD_DIR`	Directory containing the Adobe PPD (printer descriptions) files.
`KLGPF_GLOBAL_PRINTERS_FILE`	File that associates printer with PPD files.
`KLGPF_GLOBAL_TFM_DIR`	Directory containing HP TFM (font metric) files.

KlgPfObjectGetClass()

Returns the specified object's class pointer. This class pointer is defined in the object's public header file. This can be used to determine an object's type but it is easier to use the *KlgPfObjectIs* method (for example, *KlgPfObjectIsColor()*) specified for each object.

```
KlgPfObjectClass
KlgPfObjectGetClass(
   KlgPfObject   object
   )
```

KlgPfObjectGetParent()

Returns the parent of the specified object.

```
KlgPfObject
KlgPfObjectGetParent(
   KlgPfObject   object
   )
```

Page Number Macros

A page number will be automatically generated if *KlgPfFlowPrint()* or *KlgPfFramePrint()* detects an embedded macro in the passed string. Macros are embedded with an @ sign followed by one of the following number-style flags:

Flag	Function
P	The Document's `PageNumber` attribute as an Arabic page number (1, 2, 3...).
r	The Document's `PageNumber` attribute as a Roman page number, lower case (i, ii, iii...).

R	The Document's `PageNumber` attribute as a Roman page number, upper case (I, II, III...).
T	The Document's `PageCount` attribute (that is, the total number of pages in the document).
t	The total page numbers in a section (specifically, it's the number of pages since resetting the Document object's `PageNumber` attribute).

R

Data Types

This appendix lists the PageFormatter data types in alphabetical order. The C language definition of structures is also provided.

KlgPfBorderType

Enumeration used by *KlgPfFrameSetBorderType()* to specify a Frame object's border type.

```
KLGPF_BORDER_NONE
KLGPF_BORDER_PLAIN
```

KlgPfCallbackReason

Enumeration used in callbacks to specify the callback reason.

```
KLGPF_REASON_LIST_ADD
KLGPF_REASON_LIST_DELETED
KLGPF_REASON_LIST_MOVE
KLGPF_REASON_LIST_PENDING_DELETE
KLGPF_REASON_LIST_SET
KLGPF_REASON_FRAME_NEW
KLGPF_REASON_FRAME_FLOW
KLGPF_REASON_FRAME_END
KLGPF_REASON_PAGE_NEW
KLGPF_REASON_PAGE_FLOW
KLGPF_REASON_PAGE_END
KLGPF_REASON_SECTION_NEW
KLGPF_REASON_SECTION_FLOW
KLGPF_REASON_OUTPUT_NEW
KLGPF_REASON_OUTPUT_PAGE
KLGPF_REASON_OUTPUT_END
```

KlgPfCallbackProc

Procedure that make use of the information generated by a callback.

```
typedef void (*KlgPfCallbackProc)(
   XtPointer   cbs,
   XtPointer   call_data
   );
```

KlgPfCallbackRecord

Structure defining an object's callback attribute.

```
typedef struct {
   KlgPfCallbackProc     proc;
   XtPointer             call_data;
} KlgPfCallbackRecord;
```

KlgPfCellAlignment

Enumeration used by *KlgPfTableCellSetAlignment()* to specify vertical alignment of text in a cell.

```
KLGPF_CELL_ALIGNMENT_NONE
KLGPF_CELL_ALIGNMENT_TOP
KLGPF_CELL_ALIGNMENT_CENTER
KLGPF_CELL_ALIGNMENT_BOTTOM
```

KlgPfErrorStruct

Structure passed to the current error handler on error. By default this is *KlgPfError()*.

```
typedef struct {
   int           error_level;    /* read only */
   String        location;       /* read only */
   String        message;        /* read only */
   Boolean       exit;
} KlgPfErrorStruct;
```

KlgPfFillRule

Enumeration used by *KlgPfDrawFilledPolyline()* drawing method to determine how to fill overlapping areas.

```
KLGPF_FILL_RULE_NON_ZERO_WINDING
KLGPF_FILL_RULE_EVEN_ODD
```

KlgPfFillType

Enumeration used by *KlgPfTabSetFillType()* to specify a characters pattern used to fill the white space after the tab.

```
KLGPF_TAB_NO_FILL
KLGPF_TAB_FILL_WITH_UNDERSCORE
KLGPF_TAB_FILL_WITH_ELLIPSES
```

KlgPfFlushPolicy

Enumeration used by *KlgPfDocSetFlushPolicy()* to specify when a Page object should be destroyed.

```
KLGPF_FLUSH_NEVER
KLGPF_FLUSH_ON_OUTPUT
```

KlgPfFontStyle

Enumeration used by *KlgPfStyleSetFontStyle()*.

```
KLGPF_FONT_NORMAL
KLGPF_FONT_BOLD
KLGPF_FONT_ITALIC
KLGPF_FONT_BOLDITALIC
```

KlgPfFrameCallbackStruct

Structure passed to a callback procedure on generation of a Frame callback.

```
typedef struct {
   KlgPfCallbackReason   reason;
   KlgPfObject           frame;
   KlgPfObject           document;
   KlgPfObject           printer;
} KlgPfFrameCallbackStruct;
```

KlgPfGlobalType

Enumeration used by *KlgPfGlobalGet()*.

```
KLGPF_GLOBAL_AFM_DIR
KLGPF_GLOBAL_PPD_DIR
KLGPF_GLOBAL_PRINTERS_FILE
KLGPF_GLOBAL_TFM_DIR
```

KlgPfImageAlignment

Enumeration used by the Image object.

```
KLGPF_IMAGE_BOTTOM
KLGPF_IMAGE_CENTER
KLGPF_IMAGE_TOP
KLGPF_IMAGE_BASELINE
```

KlgPfImageType

Enumeration used by *KlgPfFlowEmbedImage(), KlgPfFlowFloatImage()*, *KlgPfFrameEmbedImage()*, *KlgPfFrameFloatImage()*, and *KlgPfFramePlaceImageXY()* to specify whether the image is an EPS, XPM, JPEG or GIF file.

```
KLGPF_IMAGE_TYPE_XPM
KLGPF_IMAGE_TYPE_EPS
KLGPF_IMAGE_TYPE_JPEG
KLGPF_IMAGE_TYPE_GIF
```

KlgPfIOTypeRec

Structure defining an `KlgPfIOType` type.

```
typedef struct {
   KlgPfIOOpenProc       open;
   KlgPfIOGetLineProc    get_line;
   KlgPfIOPutLineProc    put_line;
   KlgPfIOClose          close;
} KlgPfIOTypeRec;
```

KlgPfLayoutStatus

Enumeration used to specify the relative readiness of the frame or page to be printed

```
KLGPF_LAYOUT_LEVEL1
KLGPF_LAYOUT_LEVEL2
KLGPF_LAYOUT_COMPLETE
```

KlgPfOutputCallbackStruct

Structure passed to a callback procedure on generation of a printer callback.

```
typedef struct {
   KlgPfCallbackReason   reason;
```

```
   KlgPfObject          document;
   KlgPfObject          printer;
} KlgPfOutputCallbackStruct;
```

KlgPfOutputPolicy

Enumeration used to specify when rendered pages should be physically printed.

```
KLGPF_OUTPUT_AS_REQUESTED
KLGPF_OUTPUT_ASAP
```

KlgPfPageCallbackStruct

Structure passed to a callback procedure on generation of a page callback.

```
typedef struct {
   KlgPfCallbackReason   reason;
   KlgPfObject          page;
   KlgPfObject          document;
   KlgPfObject          printer;
} KlgPfPageCallbackStruct;
```

KlgPfPageType

Enumeration used to differentiate a printable Page object from a template, reference, or deleted Page object.

```
KLGPF_PAGE_ACTUAL
KLGPF_PAGE_REFERENCE
KLGPF_PAGE_TEMPLATE
KLGPF_PAGE_DELETED
```

KlgPfParagraphAlignment

Enumeration used to specify paragraph alignment.

```
KLGPF_USING_TABS
KLGPF_ALIGN_LEFT
KLGPF_ALIGN_CENTER
KLGPF_ALIGN_RIGHT
KLGPF_JUSTIFY
```

KlgPfPoint

Structure defining a point.

```
typedef struct {
   double               x, y;
} KlgPfPoint;
```

KlgPfPrinterTypeEnum

Enumeration used to specify whether a Printer object is a PostScript printer, a Acrobat PDF printer, an HP PCL printer or X window device.

```
KLGPF_PRINTER_TYPE_X11
KLGPF_PRINTER_TYPE_PS
KLGPF_PRINTER_TYPE_PCL
KLGPF_PRINTER_TYPE_ACROBAT
```

KlgPfRectangle

Structure defining Page and Frame object dimensions, relative position, and unit of measurement (for example, KLGPF_UNIT_INCHES).

```
typedef struct {
   double        x, y, width, height;
   KlgPfUnit     unit;
} KlgPfRectangle;
```

KlgPfSectionCallbackStruct

Structure passed to a callback procedure on generation of a section break.

```
typedef struct {
   KlgPfCallbackReason   reason;
   KlgPfObject           document;
   KlgPfObject           printer;
} KlgPfSectionCallbackStruct;
```

KlgPfTableBorderMode

Enumeration used to specify border mode.

```
KLGPF_TABLE_MODE_INTERNAL_BORDERS
KLGPF_TABLE_MODE_EXTERNAL_BORDERS
```

KlgPfTableBorderLocation

Enumeration used to specify border location.

```
KLGPF_TABLE_BORDER_NONE
KLGPF_TABLE_BORDER_TOP
KLGPF_TABLE_BORDER_BOTTOM
KLGPF_TABLE_BORDER_LEFT
KLGPF_TABLE_BORDER_RIGHT
KLGPF_TABLE_BORDER_ROW
KLGPF_TABLE_BORDER_COLUMN
KLGPF_TABLE_BORDER_HEADER
KLGPF_TABLE_BORDER_ALL
KLGPF_TABLE_BORDER_INVALID
```

KlgPfTableDirectionPriority

Enumeration used to specify whether Column objects or Row objects should have priority when intersecting backgrounds and borders are drawn.

```
KLGPF_TABLE_PRIORITY_ROW
KLGPF_TABLE_PRIORITY_COLUMN
```

KlgPfTableOverflowMode

Enumeration used to specify the table overflow mode.

```
KLGPF_TABLE_OVERFLOW_CLIP_COLUMNS
KLGPF_TABLE_OVERFLOW_WRAP_COLUMNS
```

KlgPfTableRuleType

Enumeration used to specify the table rule type.

```
KLGPF_TABLE_RULE_NONE
KLGPF_TABLE_RULE_SINGLE
KLGPF_TABLE_RULE_DOUBLE
KLGPF_TABLE_RULE_BROKEN
```

KlgPfTabType

Enumeration used to specify tab alignment.

```
KLGPF_TAB_RIGHT
KLGPF_TAB_LEFT
KLGPF_TAB_CENTER
KLGPF_TAB_DECIMAL
```

KlgPfTextPosition

Enumeration used to specify whether text is rendered above, below, or on the baseline.

```
KLGPF_FONT_ON_BASELINE
KLGPF_FONT_SUBSCRIPT
KLGPF_FONT_SUPERSCRIPT
```

KlgPfTextSymbolType

This enumeration is for internal PageFormatter use only.

```
KLGPF_TEXT_SYMBOL_TAB
KLGPF_TEXT_SYMBOL_SOFT_LINE_BREAK
KLGPF_TEXT_SYMBOL_HARD_LINE_BREAK
```

KlgPfUnit

Enumeration used to specify unit of measurement.

```
KLGPF_UNIT_UNKNOWN
KLGPF_UNIT_CM
KLGPF_UNIT_INCHES
KLGPF_UNIT_POINTS
```

S

Demos and Examples

Examples ■ *Demos*

The easiest way to familiarize yourself with PageFormatter is by running the demo programs. PageFormatter includes both examples and demos. Examples are short programs that illustrate specific features. Demos are more complete applications that illustrate several features working together. Most common programming tasks are covered in the examples and demos.

S.1 Examples (located in $KLGPRHOME/src/pf/examples)

PS, Acrobat, PCL Example	X11 Example	Description
email	xemail	A simple example that renders text to a single frame, changes font style, and aligns text with tabs.
simple_filter	xpreview	An example of how *KlgPfFilterSimple()* can be used to quickly flow text.
style_pf	xstyle_pf	A self-referential example of how to change various Style attributes.
traviata	xtraviata	An example demonstrating PageFormatter's use of multiple frames and printing non-English characters.
table_border	xtable_border	An example showing different table border styles.
table_color	xtable_color	An example of how to set colors in tables.
table_header	xtable_header	An example showing how to add a table header.
table_image	xtable_image	An example of how to add images to table cells.

PS, Acrobat, PCL Example	X11 Example	Description
table_margin	xtable_margin	An example of how to add margins to a table.
table_simple	xtable_simple	A simple example demonstrating how to use tables in PageFormatter.
table_span	xtable_span	An example of how to span table cells.
table_style	xtable_style	An example showing how to set different table styles.

S.2 Demos

S.2.1 Financial ($KLGPRHOME/src/pf/demos/financial)

A demonstration on how PageFormatter functionality may be applied to financial data.

S.2.2 Realestate ($KLGPRHOME/src/pf/demos/realestate)

Demonstrates using PageFormatter with a list of data records.

11.2.7 Reports ($KLGPRHOME/src/pf/demos/reports)

Demonstrates a variety of financial reports.

11.2.8 Webprint ($KLGPRHOME/src/pf/demos/webprint)

Demonstrates a web-based printing engine.

S.2.3 Parser ($KLGPRHOME/src/pf/demos/parser)

Parser has two versions: xparser and parser. Xparser allows you to generate an X document in the previewer (xparser) or a PostScript file (.ps) from a document marked up with formatting tags. Parser will only generate a PostScript, Acrobat or HP PCL file.

The parser comes with several examples, demonstrating the range of tags. Each demonstration consists of three files: A data (`.dat`) file which contains the text and tags, a similarly named template (`.tmpl`) file which contains the page template information, and a similarly named sample set (`.smpl`) file which can be used with xparser's **File | Open** command to automatically load the examples' `.dat` and `.tmpl` files.

Running the Parser

```
parser  [-p][-l][-o ps_file][-s smpl_file][-t tmpl_file][-i data_file]
xparser [-x][-p][-l][-s smpl_file][-t tmpl_file][-i data_file]
```

`-acro`	Flows to an Acrobat (.PDF) file.
`-l`	Generate a logfile for debugging purposes
`-p`	Flow the formatting tags
`-pcl`	Flows to an HP PCL file.
`-ps`	Flows to a PostScript file.
`-s <file>`	Select a sample set file
`-t <file>`	Select a template file
`-i <file>`	Select a data file
`parser only: -o <file>`	Name the generated postscript file
`xparser only: -x`	Flow an x based document, instead of postscript

Tags

The parser interprets the following tags and calls equivalent PageFormatter methods. If you wish to render a tag instead of having the parser interpret the tag, begin it with a backslash (\). You do not, however, have to begin a less than sign (<) with a backslash if you simply want to render it outside of a valid tag.

<align *type*>

Set paragraph alignment type.

Method Equivalent

KlgPfStyleSetAlignment()

Parameters

type: `LEFT`, `RIGHT`, `CENTER`, or `JUSTIFY`.

<columnf>

Advances the flow to the top of the next column. If there is no next column, a frame feed is called.

Method Equivalent

KlgPfFlowColumnFeed()

<docbegin>

Begin a document. Your .dat file must have `<docbegin>` before any text you wish to render in the main flow.

Method Equivalent
KlgPfDocBegin()

<docend>

End a document. This must be the last tag in your data file. Anything after this tag is parsed but ignored.

Method Equivalent
KlgPfDocEnd()

<embimage *bounding_box_status alignment image_type image width height*>

Embed an EPS, GIF, JPEG or XPM Image.

Method Equivalent
KlgPfFlowEmbedImage()

Parameters

bounding_box_status: This parameter can take one of the following values: `box_on` or `box_off`. `box_on` tells the parser to turn on the bounding box for the image being rendered. `box_off` tell the parser to turn off the bounding box for the image being rendered.

alignment: `TOP`, `BOT`, `CEN`, or `BAS`.

image_type: `EPS`, `GIF`, `JPEG` or `XPM`.

image: an EPS, GIF, JPEG or XPM file.

width, height: *height* and *width* specify the displayed (scaled) size of an image.

<fltimage *bounding_box_status image_type image width height*>

Float an EPS, GIF, JPEG or XPM Image.

Method Equivalent
KlgPfFlowFloatImage()

Parameters

bounding_box_status: This parameter can take one of the following values: `box_on` or `box_off`. `box_on` tells the parser to turn on the bounding box for the image being rendered. `box_off` tell the parser to turn off the bounding box for the image being rendered.

image_type:`EPS`, `GIF`, `JPEG` or `XPM`.

image: an EPS, GIF, JPEG or XPM file.

width, height: *height* and *width* specify the displayed (scaled) size of the EPS image.

<flushpol *policy*>

Set flush policy

Method Equivalent
KlgPfDocSetFlushPolicy()

Parameters

policy: `NEVER` or `ON_OUTPUT`.

<fontcolor *color_string*>

Sets font color.

Method Equivalent

KlgPfStyleSetColor()

Parameters

color_string: A color string (for example, `<fontcolor red>`).

<fontfam *font_family_name*>

Sets the font family.

Method Equivalent

KlgPfStyleSetFontFamily()

Parameters

font_family_name: A font family (for example, `<fontfam helvetica>`).

<fontsize *int*>

Sets the font's point size.

Method Equivalent

KlgPfStyleSetFontSize()

Parameters

int: An integer value greater than 0.

<fontstyle *font_style*>

Sets the font to print as normal, bold, italics, or bold-italics.

Method Equivalent

KlgPfStyleSetFontStyle()

Parameters

font_style: `NORMAL`, `BOLD`, `ITALIC`, or `BOLDITALIC`.

<formf>

Generates a new page based on the page template defined by the current Page object's `FlowPageTemplate` attribute. The flow is transferred to the first frame on the new page.

Method Equivalent

KlgPfFlowFormFeed()

<framef>

Generates a frame feed and transfers the flow to the next frame indicated by the Page object's `FlowFrameList` attribute.

Method Equivalent

KlgPfFlowFrameFeed()

<frcolumnf *template frame*>

Advances the flow to the top of the next column. If there is no next column, a frame feed is called.

Method Equivalent

KlgPfFlowColumnFeed()

Parameters

template: The template in which you wish to generate the column feed. Required if used before `<docbegin>`. Should not be included if used after `<docbegin>`.

frame: The frame in which you wish to generate the column feed.

<frembimage *bounding_box_status template frame alignment image_type image width height*>

Draws the specified EPS, GIF, JPEG or XPM image as an inline graphic in the specified frame.

Method Equivalent

KlgPfFrameEmbedImage()

Parameters

bounding_box_status: This parameter can take one of the following values: `box_on` or `box_off`. `box_on` tells the parser to turn on the bounding box for the image being rendered. `box_off` tell the parser to turn off the bounding box for the image being rendered.

template: The template in which you wish to generate the image. Required if used before `<docbegin>`. Should not be included if used after `<docbegin>`.

frame: The frame in which you wish to generate the image.

alignment: `TOP`, `CEN`, `BOT`, or `BAS`.

image_type: `EPS, GIF, JPEG or XPM`.

image: an EPS, GIF, JPEG or XPM file.

width, height: *height* and *width* specify the displayed (scaled) size of the EPS image.

<frfltimage *bounding_box_status template frame image_type image width height*>

Draws the specified EPS or XPM image on its own line in the specified frame.

Method Equivalent

KlgPfFrameFloatImage()

Parameters

bounding_box_status: This parameter can take one of the following values: `box_on` or `box_off`. `box_on` tells the parser to turn on the bounding box for the image being rendered. `box_off` tell the parser to turn off the bounding box for the image being rendered.

template: The template in which you wish to generate the image. Required if used before `<docbegin>`. Should not be included if used after `<docbegin>`.

frame: The frame in which you wish to generate the image.

image_type: `EPS, GIF, JPEG or XPM`.

image: an EPS, GIF, JPEG or XPM file.

width, height: *height* and *width* specify the displayed (scaled) size of the EPS image.

<frlinef *template frame*>

Generates a line feed in the specified frame.

Method Equivalent

KlgPfFrameLineFeed()

Parameters

template: The template in which you wish to generate the line feed. Required if used before `<docbegin>`. Should not be included if used after `<docbegin>`.

frame: The frame in which you wish to generate the line feed.

<frprint *template frame "text"*>

Prints text to the indicated template and frame.

Method Equivalent

KlgPfFramePrint()

Parameters

template: The template in which you wish to generate the line feed. Required if used before `<docbegin>`. Should not be included if used after `<docbegin>`.

frame: The frame in which you wish to generate the line feed.

text: The text you wish to render in the frame (for example, `<frprint my_template footer "Page @p">`).

<frtabf *template frame*>

Generates a tab in the specified frame.

Method Equivalent

KlgPfFrameTab()

Parameters

template: The template in which you wish to generate the tab. Required if used before `<docbegin>`. Should not be included if used after `<docbegin>`.

frame: The frame in which you wish to generate the tab.

<indfirst *double*>

Sets the first line indent.

Method Equivalent

KlgPfStyleSetIndentFirstLine()

Parameters

double: Indent amount.

<indleft *double*>

Sets the subsequent left line indent.

Method Equivalent

KlgPfStyleSetIndentLeft()

Parameters

double: Indent amount.

<indright *double*>

Sets the right line indent.

Method Equivalent

KlgPfStyleSetIndentRight()

Parameters

double: Indent amount.

<linef>

Generates a line feed.

Method Equivalent

KlgPfFlowLineFeed()

<linespace *double*>

Sets the line spacing amount.

Method Equivalent

KlgPfStyleSetLineSpacing()

Parameters

double: amount of space between lines.

<memorymax *int*>

Specifies the maximum amount of memory that should be used for the rendering purposes.

Method Equivalent

int: an integer value.

<outputpol *policy*>

Sets the output policy.

Method Equivalent

KlgPfDocSetOutputPolicy()

Parameters

policy: `AS_REQUESTED` or `ASAP`.

<paraspace *double*>

Sets after paragraph spacing.

Method Equivalent

KlgPfStyleSetParagraphSpacing()

Parameters

double: a double value.

<secbrk>

Generates a section break.

Method Equivalent

KlgPfFlowSectionBreak()

<tabf>

Generates a tab.

Method Equivalent

KlgPfFlowTab()

<tabfill *fill_type*>

Sets the tab fill.

Method Equivalent

KlgPfTabSetFillType()

Parameters

fill_type: `NO_FILL`, `ELLIPSE`, or `UNDERSCORE`.

<tabsbyvaluelist *[x1 x2 x3 ...]*>

Sets one or more tabs.

Method Equivalent

KlgPfStyleSetTabsByValueList()

Parameters

[x1 x2 x3 ...]: an array of values (for example, `<tabsbyvaluelist [2.3]>` or `<tabsbyvaluelist [1 2.3 4.5 5]>`).

<tabsequallyspaced *start increment ntabs*>

Sets a series of tabs, spaced at equal intervals.

Method Equivalent

KlgPfStyleSetTabsEquallySpaced()

Parameters

start: The x position of the first Tab object.

increment: How far apart the Tab stops should be set.

ntabs: The number of Tab objects to create.

<tabtype *type*>

Sets the tab type.

Method Equivalent

KlgPfTabSetTabType()

Parameters

type: `LEFT`, `RIGHT`, `DECIMAL`, or `CENTER`.

<textpos *alignment*>

Sets text to render as superscript, subscript, or on the baseline.

Method Equivalent

KlgPfTabSetTabType()

Parameters

type: `LEFT`, `RIGHT`, `DECIMAL`, or `CENTER`.

<underline *on_off*>

Turns underlining on or off.

Method Equivalent

KlgPfStyleSetUnderline()

Parameters

on_off: `ON` or `OFF`.

Example .dat File

A complete (albeit minimal) tagged data file might look like:

```
<docbegin>
<fontstyle BOLD>
Danger!
<fontstyle NORMAL>
Come quick.
<linef>
HMS Titanic
<docend>
```

The above would produce:

Danger! Come quick.
HMS Titanic

Table Tags

<cell>

Signifies the start of a new table cell.

Method Equivalent

None

<cellalign alignment>

Sets the table cell's alignment.

Method Equivalent

KlgPfTableCellSetAlignment()

Parameters

alignment: `TOP`, `BOTTOM`, `CENTER`, `NONE`

<cellbackcolor color>

Method Equivalent

KlgPfTableCellSetBackgroundColor()

Parameters

color: A color string. For example: `<cellbackcolor red>`.

<cellborder location color width rule_type spacing>

Method Equivalent

KlgPfTableCellSetBorderStyle()

Parameters

location: `NONE`, `TOP`, `BOTTOM`, `LEFT`, `RIGHT`, `ROW`, `COLUMN`, `HEADER`, `ALL`

color: A color string. For example: `<cellborder TOP red .02 BROKEN .04>`.

width: The border's width, in document units.

rule_type: `NONE`, `SINGLE`, `BROKEN`, `DOUBLE`, `BROKEN`.

spacing: if *rule_type* is `BROKEN`, *spacing* indicates the length of the dashes and the white space. If *rule_type* is `DOUBLE`, *spacing* is the amount of white space between the double rules.

<cellembimage alignment image_type bounding_box_status image width height>

Embed an EPS, GIF, JPEG or XPM Image in the cell.

Method Equivalent

KlgPfFrameEmbedImage()

Parameters

alignment: `TOP`, `BOT`, `CEN`, or `BAS`.

bounding_box_status: This parameter can take one of the following values: `box_on` or `box_off`. `box_on` tells the parser to turn on the bounding box for the image being rendered. `box_off` tell the parser to turn off the bounding box for the image being rendered.

image_type: `EPS, GIF, JPEG` or `XPM`.

image: an EPS, GIF, JPEG or XPM file.

width, height: *height* and *width* specify the displayed (scaled) size of the EPS image.

<cellinef>

Performs a line feed in the current cell of a table.

<cellfltimage bounding_box_status image_type image width height>

Float an EPS, GIF, JPEG or XPM Image in the cell.

Method Equivalent

KlgPfFrameFloatImage()

Parameters

bounding_box_status: This parameter can take one of the following values: `box_on` or `box_off`. `box_on` tells the parser to turn on the bounding box for the image being rendered. `box_off` tell the parser to turn off the bounding box for the image being rendered.

image_type: `EPS, GIF, JPEG or XPM`.

image: an EPS, GIF, JPEG or XPM file.

width, height: *height* and *width* specify the displayed (scaled) size of the EPS image.

<cellheight height>

Method Equivalent

KlgPfTableCellSetHeight()

Parameters

height: the cell's height, in document units.

<cellmarginbottom width>

Sets the cell's bottom margin.

Method Equivalent

KlgPfTableCellSetMarginBottom()

Parameters

width: margin width, in document units.

<cellmarginleft width>

Sets the cell's left margin.

Method Equivalent

KlgPfTableCellSetMarginLeft()

Parameters

width: margin width, in document units.

<cellmarginright width>

Sets the cell's right margin.

Method Equivalent

KlgPfTableCellSetMarginRight()

Parameters

width: margin width, in document units.

<cellmargintop width>

Sets the cell's top margin.

Method Equivalent

KlgPfTableCellSetMarginTop()

Parameters

width: margin width, in document units.

<colalign colnum alignment>

Sets the alignment of the cells in the column.

Method Equivalent

KlgPfTableColumnSetAlignment()

Parameters

colnum: index of the desired column.

alignment: `TOP`, `BOTTOM`, `CENTER`, `NONE`

<colbackcolor colnum color>

Sets the columns background color.

Method Equivalent

KlgPfTableColumnSetBackgroundColor()

Parameters

colnum: index of the desired column.

color: A color string. For example: `<colbackcolor 2 red>`.

<colborder colnum color width rule_type spacing>

Sets the column's border.

Method Equivalent

KlgPfTableColumnSetBorder()

Parameters

colnum: index of the desired column.

color: A color string. For example: `<colborder 1 red .02 DOUBLE .04>`

width: border width, in document units.

rule_type: `NONE`, `SINGLE`, `BROKEN`, `DOUBLE`, `BROKEN`

spacing: if *rule_type* is `BROKEN`, *spacing* indicates the length of the dashes and the white space. If *rule_type* is `DOUBLE`, *spacing* is the amount of white space between the double rules.

<colmarginbottom colnum width>

Sets the bottom margin.

Method Equivalent

KlgPfTableColumnSetCellMarginBottom()

Parameters

colnum: index of the desired column.

width: margin width, in document units.

<colmarginleft colnum width>

Sets the left margin.

Method Equivalent

KlgPfTableColumnSetCellMarginLeft()

Parameters

colnum: index of the desired column.

width: margin width, in document units.

<colmarginright colnum width>

Sets the right margin.

Method Equivalent

KlgPfTableColumnSetCellMarginRight()

Parameters

colnum: index of the desired column.

width: margin width, in document units.

<colmargintop colnum width>

Sets the top margin.

Method Equivalent

KlgPfTableColumnSetCellMarginTop()

Parameters

colnum: index of the desired column.

width: margin width, in document units.

<headerbegin>

Indicates that tags following this tag apply to the header table. Terminate with `<headerend>`.

Method Equivalent

None

<headerend>

Indicates that tags following this tag no longer apply to the header table.

Method Equivalent

None

<row>

Signifies the start of a new row. All row property changes affect the current row.

Method Equivalent

None

<rowalign alignment>

Sets the row's vertical alignment.

Method Equivalent

KlgPfTableRowSetAlignment()

Parameters

alignment: `TOP`, `BOTTOM`, `CENTER`, `NONE`

<rowbackcolor color>

Sets the row's background color.

Method Equivalent

KlgPfTableRowSetBackgroundColor()

color: A color string. For example: `<rowbackcolor red>`

<rowborder color width rule_type spacing>

Method Equivalent

KlgPfTableRowSetBorder()

color: A color string. For example: `<rowborder red  0.02 SINGLE>`

width: The border's width, in document units

rule_type: `NONE`, `SINGLE`, `BROKEN`, `DOUBLE`, `BROKEN`.

spacing: if *rule_type* is `BROKEN`, *spacing* indicates the length of the dashes and the white space. If *rule_type* is `DOUBLE`, *spacing* is the amount of white space between the double rules.

<rowmarginbottom width>

Sets the row's bottom margin.

Method Equivalent

KlgPfTableRowSetCellMarginBottom()

width: margin width, in document units.

<rowmarginleft width>

Sets the row's left margin.

Method Equivalent

KlgPfTableRowSetCellMarginLeft()

Parameters

width: margin width, in document units.

<rowmarginright width>

Sets the row's right margin.

Method Equivalent

KlgPfTableRowSetCellMarginRight()

Parameters

width: margin width, in document units.

<rowmargintop width>

Sets the row's top margin.

Method Equivalent

KlgPfTableRowSetCellMarginTop()

Parameters

width: margin width, in document units.

<tablealign alignment>

Sets the table's horizontal alignment.

Method Equivalent

KlgPfTableSetAlignment()

Parameters

alignment: LEFT, CENTER, RIGHT, JUSTIFY

<tablebackcolor color>

Sets the table's background color.

Method Equivalent

KlgPfTableSetBackgroundColor()

Parameters

color: A color string. For example: `<rowborder red  0.02 SINGLE>`

<tablebegin c1 c2 c3 ...>

Creates a table with as many columns as there are supplied column widths.

Method Equivalent

KlgPfTableCreate()

Parameters

c1, c2, c3, …: are column widths in document units.

<tableborder location color width rule_type spacing>

Sets the table border.

Method Equivalent

KlgPfTableSetBorderStyle()

Parameters

location: NONE, TOP, BOTTOM, LEFT, RIGHT, ROW, COLUMN, HEADER, ALL

color: A color string. For example: `<cellborder TOP red .02 BROKEN .04>`

width: The border's width, in document units

rule_type: NONE, SINGLE, BROKEN, DOUBLE, BROKEN.

spacing: if *rule_type* is BROKEN, *spacing* indicates the length of the dashes and the white space. If *rule_type* is DOUBLE, *spacing* is the amount of white space between the double rules.

<tablecellalign alignment>

Sets the alignment of text in the table's cells.

Method Equivalent

KlgPfTableSetCellAlignment()

Parameters

alignment: TOP, BOTTOM, CENTER, NONE

<tabledirection direction>

Sets the Table object's PriorityDirection.

Method Equivalent

KlgPfTableSetPriorityDirection()

Parameters

direction: `ROW` or `COLUMN`

<tableend>

Indicates that normal document flow resumes.

Method Equivalent

KlgPfTablePrintAndDestroy()

<tablemarginbottom width>

Sets the Table cells' bottom margin.

Method Equivalent

KlgPfTableSetCellMarginBottom()

Parameters

width: margin width, in document units.

<tablemarginleft width>

Sets the Table cells' left margin.

Method Equivalent

KlgPfTableSetCellMarginLeft()

Parameters

width: margin width, in document units.

<tablemarginright width>

Sets the Table cells' right margin.

Method Equivalent

KlgPfTableSetCellMarginRight()

Parameters

width: margin width, in document units.

<tablemargintop width>

Sets the Table cells' top margin.

Method Equivalent

KlgPfTableSetCellMarginTop()

Parameters

width: margin width, in document units.

<tableoverflow mode>

Sets how page or column wrap breaks should be rendered.

Method Equivalent

KlgPfTableSetOverflowMode()

Parameters

mode: `CLIP`, `WRAP`

<tablespancells row col numrows numcols>

Sets a cell span.

Method Equivalent

KlgPfTableSpanCells()

Parameters

row: The row index of the span-originating cell.

col: The column index of the span-originating cell.

numrows: The number of rows down the cell should span.

numcols: The number of columns to the right the cell should span.

Example .DAT File using Table Tags

```
<docbegin>
<tablebegin 2 2 2>
<row>
<tablespancells 1 1 1 3>
<cell>
<cellbackcolor Gray50>
<fontstyle BOLD>
Simple Table
<fontstyle NORMAL>
<row>
<cell>
I am a cell at (0,0)
<cell>
I am a cell at (0,1)
<cell>
I am a cell at (0,2)
<row>
<cell>
I am a cell at (1,0)
<cell>
I am a cell at (1,1)
<cell>
I am a cell at (1,2)
<row>
<cell>
I am a cell at (2,0)
<cell>
I am a cell at (2,1)
<cell>
I am a cell at (2,2)
<tableend>
<docend>
```

The above would produce:

Simple Table		
I am a cell at (0,0)	I am a cell at (0,1)	I am a cell at (0,2)
I am a cell at (1,0)	I am a cell at (1,1)	I am a cell at (1,2)
I am a cell at (2,0)	I am a cell at (2,1)	I am a cell at (2,2)

Index

.AFM file 7
.PPD file 7

A

Adding Columns to a Frame 130
Adobe Acrobat PDF 113
AFM Files 73
Alignment 75, 243, 260, 262, 264, 265
ANSI C, compatibility with PageFormatter 8
Attributes 21
Authorizing a Program 8

B

BackgroundColor 261, 262, 264, 265
Bold 171
BoldItalic 171
Border Color 52
Border Type 52
Border Width 52
Border, Adding to a Frame in Template Builder 130
BorderBottom 261, 264, 265
BorderColor 41, 175
BorderColumn 265
BorderHeader 265
BorderLeft KlgPfObject Single border, Width 0.0, Black 266
BorderMode 266
BorderRight 261, 263, 266
BorderRow 266
BorderTop 266
BorderType 40, 175
BorderWidth 40, 176
Bullet, Creating 82

C

C++
  compatibility with PageFormatter 8
Callback Structure 32
Callbacks 32, 114, 126
Cell Height 99
CellAlignment 98, 267
CellList 264
CellMarginBottom 263, 264, 267
CellMarginLeft 263, 264, 267
CellMarginRight 263, 264, 267
CellMarginTop 263, 265, 267
CellStyle 97, 261, 263, 265, 267
Changing Font Appearance 71
Collate 221
Collating 110
Color 40, 176, 244, 260
  Monochrome 84
  Predefined Strings 84
  Setting RGB Individually 83
Color Object 82
ColumnCount 40, 53, 176
ColumnList 267
Columns 53
Columns, Adding to a Frame 130
ColumnSpacing 40, 53, 176
ColumnWidth 54, 176
CompareProc 202
Compiling and Authorizing a Program 8
Copying a Style 68
Create Methods 22
Creating a List 119
Creating a Style Object 68
Current Style 30
CurrentDocument 221
CurrentDrawStyle 62, 148
CurrentFlow 147
CurrentFrame 147
CurrentPage 117, 147, 234
CurrentStyle 69, 147

D

Data Type Listing 303, 309
DefaultValue 237
Defining a Page Template 38
Defining Procedures 33, 115, 126
Demo Programs 310
Description 237
Destroy Methods 22
Destroying a List 119
Destroying a Page Object 47
DestroyProc 202
Direction Attribute 107
Direction Priority 107
DirectionPriority 268
Display 143
DisplayX11 221

Document 235
Document Object 27
Document Units 22, 150
Drawable 144
Drawing Tools 60
DrawStyle Object 61
DrawStyleList 62, 148
Duplicating a List 119

E

Embedded Graphics 85
EPS 85
Error Levels 25
Error Structure 25
Errors
  Methods 26
  Returns on Error 26
Errors Handling 24
Executable, Compiling and Authorizing 8

F

Filename 113, 114, 222
Fill Type 80
FillRule Attribute 62
FillType 255
Financial Demo 310
First Line Indent 76
Floating Images 87
Flow 30
FlowFrameList 40, 41, 54, 213
FlowFrameList list 32
Flowing Images 85
FlowPageTemplate 41, 54, 214
FlowSectionTemplate 41, 214
FlushPolicy 47, 148
Font 244
FontFamily 72, 73, 244
FontFamilyList 222
FontList 73, 222
Fonts 71
  Font Family 72
  Lists 73
  Size 74
  Style 74
  Underlining 75
FontSize 74, 244
FontStyle 73, 244
Footer Frame 19, 49
Frame 261
  Border Color 52
  Border Type 52
  Borders 51
  ColumnCount 53
  Columns 53
  ColumnSpacing 53
  ColumnWidth 54
  Footer 19, 49, 54
  Header 54
  Height 51
  MarginBottom 53
  MarginLeft 53
  MarginRight 53
  Margins 52
  MarginTop 53
  Printing To 55
  Setting Attributes 51
  Width 51
  Y Coordinate 51
Frame Color, Setting in Template Builder 129
Frame Object 19, 49
Frame Size, Setting in Template Builder 130
Frame X Coordinate 51
Frame, Adding in Template Builder 129
FrameCallback 33, 34, 148
FrameList 57, 214
Frames
  Multiple 54

G

Get Methods 21
GIF 85

H

Header Frame 19, 49
Height 99, 235, 261
HP TFM file 7

I

Images 85
  Embedded Graphics 85
  Floating 87
IndentFirstLine 69, 244
IndentLeft 244
IndentRight 244
Indents 76
Index 329
Introduction 1
  Typographical Conventions 1
Italic 171
ItemCount 202
Items 202
ItemSize 202

J

JPEG 85

K

KlgGlobalSet() 7
KLGPF_ALIGN_CENTER 76, 97, 243, 265
KLGPF_ALIGN_JUSTIFY 76
KLGPF_ALIGN_LEFT 76, 97, 243, 265
KLGPF_ALIGN_RIGHT 76, 97, 243, 265
KLGPF_BORDER_NONE 52, 175
KLGPF_BORDER_PLAIN 52, 176
KLGPF_CELL_ALIGNMENT_BOTTOM 267
KLGPF_CELL_ALIGNMENT_CENTER 267
KLGPF_CELL_ALIGNMENT_NONE 267
KLGPF_CELL_ALIGNMENT_TOP 267
KLGPF_FLUSH_NEVER 47, 148, 155
KLGPF_FLUSH_ON_OUTPUT 47, 148, 155
KLGPF_FONT_BOLD 74, 244
KLGPF_FONT_BOLDITALIC 74, 244
KLGPF_FONT_ITALIC 74, 244
KLGPF_FONT_NORMAL 74, 244
KLGPF_FONT_ON_BASELINE 74, 245
KLGPF_FONT_SUBSCRIPT 74, 245
KLGPF_FONT_SUPERSCRIPT 74, 245
KLGPF_IMAGE_BASELINE 86, 160, 183
KLGPF_IMAGE_BOTTOM 86, 160, 183
KLGPF_IMAGE_CENTER 86, 160, 183
KLGPF_IMAGE_TOP 86, 160, 183
KLGPF_JUSTIFY 97, 244, 265
KLGPF_LAYOUT_COMPLETE 114, 176, 214
KLGPF_LAYOUT_LEVEL1 176, 214
KLGPF_LAYOUT_LEVEL2 176, 214
KLGPF_LIST_ITEM_NOT_FOUND 122, 124
KLGPF_OUTPUT_AS_REQUESTED 114, 149
KLGPF_OUTPUT_ASAP 114, 149
KLGPF_PAGE_ACTUAL 214
KLGPF_PAGE_REFERENCE 214
KLGPF_PAGE_TEMPLATE 43, 215
KLGPF_PRINTER_TYPE_PS 223
KLGPF_PRINTER_TYPE_X11 223
KLGPF_REASON_FRAME_END 34, 148
KLGPF_REASON_FRAME_FLOW 34, 148
KLGPF_REASON_FRAME_NEW 34, 148
KLGPF_REASON_OUTPUT_END 115
KLGPF_REASON_OUTPUT_NEW 114
KLGPF_REASON_OUTPUT_PAGE 115
KLGPF_REASON_PAGE_END 34, 149
KLGPF_REASON_PAGE_FLOW 34, 149
KLGPF_REASON_PAGE_NEW 34, 149
KLGPF_REASON_SECTION_FLOW 35, 150
KLGPF_REASON_SECTION_NEW 35, 150
KLGPF_TAB_CENTER 81, 256
KLGPF_TAB_DECIMAL 81, 256
KLGPF_TAB_FILL_WITH_ELLIPSES 15, 80, 255
KLGPF_TAB_FILL_WITH_UNDERSCORE 80, 255
KLGPF_TAB_LEFT 80, 256
KLGPF_TAB_NO_FILL 80, 255
KLGPF_TAB_RIGHT 80, 256
KLGPF_TABLE_MODE_EXTERNAL_BORDERS 266
KLGPF_TABLE_MODE_INTERNAL_BORDERS 266
KLGPF_UNIT_CM 150, 223
KLGPF_UNIT_INCHES 150, 223
KLGPF_UNIT_POINTS 150, 223
KLGPF_USING_TABS 243
KlgPfBorderLocation 307
KlgPfBorderMode 307
KlgPfBorderType 303
KlgPfCallbackProc 303
KlgPfCallbackReason 303
KlgPfCallbackRecord 148, 149, 150, 304
KlgPfColorCopy() 136
KlgPfColorCreate() 83, 136
KlgPfColorCreateFromString() 136
KlgPfColorCreateGrey() 84, 137
KlgPfColorCreateRGB() 138
KlgPfColorDestroy() 83, 138
KlgPfColorGetBlue() 138
KlgPfColorGetGreen() 138
KlgPfColorGetGrey() 138
KlgPfColorGetRed() 139
KlgPfColorGetRGB() 139
KlgPfColorSetBlue() 83, 139
KlgPfColorSetGreen() 83, 139
KlgPfColorSetGrey() 140
KlgPfColorSetRed() 83, 140
KlgPfColorSetRGB() 83, 140
KlgPfColorTestEqual() 141
KlgPfConvertCmToInches() 297
KlgPfConvertCmToPoints() 297
KlgPfConvertInchesToCm() 297
KlgPfConvertInchesToPoints() 298
KlgPfConvertPointsToCm() 298
KlgPfConvertPointsToInches() 77, 298
KlgPfDisplayX11Create 144
KlgPfDisplayX11Create() 144
KlgPfDisplayX11Destroy() 144
KlgPfDisplayX11GetDisplay() 144
KlgPfDisplayX11GetDrawable() 144
KlgPfDisplayX11GetWindow() 145
KlgPfDisplayX11SetDisplay() 145
KlgPfDisplayX11SetDrawable() 145
KlgPfDisplayX11SetWindow() 145
KlgPfDocBegin() 10, 151
KlgPfDocCreate() 28, 57, 151
KlgPfDocDestroy() 12, 28, 151
KlgPfDocDestroy() and KlgPfPrinterDestroy() 12
KlgPfDocEnd() 11, 28, 151
KlgPfDocGetCurrentDrawStyle() 152
KlgPfDocGetCurrentFlow() 152
KlgPfDocGetCurrentFrame() 152
KlgPfDocGetCurrentPage() 152
KlgPfDocGetCurrentStyle() 69, 152
KlgPfDocGetDrawStyleList() 152
KlgPfDocGetFlushPolicy() 153
KlgPfDocGetMemoryMax() 153
KlgPfDocGetName() 153
KlgPfDocGetOutputPolicy() 153
KlgPfDocGetPageCount() 29, 153
KlgPfDocGetPageList() 153
KlgPfDocGetPageNumber() 29, 154

KlgPfDocGetPageTemplateList() 154
KlgPfDocGetPrinter() 154
KlgPfDocGetStyleList() 154
KlgPfDocGetUnit() 154, 157
KlgPfDocSetCurrentDrawStyle() 62, 154
KlgPfDocSetCurrentFlow() 155
KlgPfDocSetCurrentStyle() 30, 155
KlgPfDocSetFlushPolicy() 155
KlgPfDocSetFrameCallback() 33, 155
KlgPfDocSetMemoryMax() 156
KlgPfDocSetOutputPolicy() 114, 156
KlgPfDocSetPageCallback() 33, 156
KlgPfDocSetPageNumber() 29, 59, 157
KlgPfDocSetPrinter() 157
KlgPfDocSetSectionCallback() 33, 157
KlgPfDocSetUnit() 22
KlgPfDocStringToDrawStyle() 158
KlgPfDocStringToStyle() 30, 57, 158
KlgPfDocStringToTemplate() 57, 158
KlgPfDrawCircle() 63, 177
KlgPfDrawFilledCircle() 178
KlgPfDrawFilledPolyline() 63, 178
KlgPfDrawFilledRectangle() 179
KlgPfDrawFilledRoundedRectangle() 179
KlgPfDrawPolyline() 180
KlgPfDrawRectangle() 180
KlgPfDrawRoundedRectangle() 64, 180
KlgPfDrawSimpleLine() 62, 181
KlgPfDrawStyleCreate() 62
KlgPfDrawStyleSetColor() 62
KlgPfDrawStyleSetLineWidth() 62
KlgPfError() 298
KlgPfErrorGetHandler() 298
KlgPfErrorGetUserData() 299
KlgPfErrorSetHandler() 299
KlgPfErrorSetUserData() 299
KlgPfErrorStruct 304
KlgPfFillRule 304
KlgPfFillType 304
KlgPfFilterSimple() 299
KlgPfFlowColumnFeed() 158
KlgPfFlowEmbedImage() 85
KlgPfFlowFloatImage() 87
KlgPfFlowFormFeed() 32, 158
KlgPfFlowFrameFeed() 32, 41, 159
KlgPfFlowHRuleLength() 159
KlgPfFlowHRulePercent() 159
KlgPfFlowImageEmbed() 159
KlgPfFlowImageFloat() 160
KlgPfFlowLineFeed() 32, 160
KlgPfFlowPrint() 11, 32, 43, 46, 160
KlgPfFlowSectionBreak() 32, 41, 161
KlgPfFlowTab() 15, 81, 161
KlgPfFlushPolicy 304
KlgPfFontCreate() 168
KlgPfFontDestroy() 168
KlgPfFontFamilyCreate() 172
KlgPfFontFamilyDestroy() 172
KlgPfFontFamilyGetBold() 172
KlgPfFontFamilyGetBoldItalic() 173
KlgPfFontFamilyGetItalic() 173
KlgPfFontFamilyGetName() 173
KlgPfFontFamilyGetNormal() 173
KlgPfFontFamilySetBold() 173
KlgPfFontFamilySetBoldItalic() 173
KlgPfFontFamilySetItalic() 174
KlgPfFontFamilySetNormal() 174
KlgPfFontGetName() 168
KlgPfFontGetSizesList() 169
KlgPfFontStyle 304
KlgPfFrameCallbackStruct 305
KlgPfFrameColumnFeed() 181
KlgPfFrameCopy() 182
KlgPfFrameCreate() 50, 182
KlgPfFrameDestroy() 182
KlgPfFrameGetBorderColor() 184
KlgPfFrameGetBorderType() 184
KlgPfFrameGetBorderWidth() 184
KlgPfFrameGetColor() 184
KlgPfFrameGetColumnCount() 185
KlgPfFrameGetColumnSpacing() 185
KlgPfFrameGetColumnWidth() 185
KlgPfFrameGetCurrentAscender() 185
KlgPfFrameGetCurrentBaseline() 185
KlgPfFrameGetCurrentDescender() 185
KlgPfFrameGetCurrentX() 185
KlgPfFrameGetCurrentY() 186
KlgPfFrameGetHeight() 186
KlgPfFrameGetLayoutStatus() 186
KlgPfFrameGetMarginBottom() 186
KlgPfFrameGetMarginLeft() 186
KlgPfFrameGetMarginRight() 186
KlgPfFrameGetMarginTop() 187
KlgPfFrameGetName() 187
KlgPfFrameGetNextTab() 187
KlgPfFrameGetParent() 187
KlgPfFrameGetWidth() 187
KlgPfFrameGetX() 187
KlgPfFrameGetY() 188
KlgPfFrameHRuleLength() 188
KlgPfFrameHRulePercent() 188
KlgPfFrameImageEmbed() 183
KlgPfFrameImagePlaceXY() 184
KlgPfFrameLineFeed() 188
KlgPfFramePrint() 55, 57, 189
KlgPfFramePrintXY() 189
KlgPfFrameSetBorderColor() 52, 189
KlgPfFrameSetBorderType() 51, 190
KlgPfFrameSetBorderWidth() 52, 190
KlgPfFrameSetColor() 190
KlgPfFrameSetColumnCount() 53, 190
KlgPfFrameSetColumnSpacing() 53, 190
KlgPfFrameSetCurrentX() 191
KlgPfFrameSetCurrentY() 191
KlgPfFrameSetHeight() 191
KlgPfFrameSetLayoutStatus() 191
KlgPfFrameSetMarginBottom() 53, 191
KlgPfFrameSetMarginLeft() 53, 192
KlgPfFrameSetMarginRight() 53, 192
KlgPfFrameSetMarginTop() 53, 192

KlgPfFrameSetWidth() 51, 192
KlgPfFrameSetX() 51, 192
KlgPfFrameSetY() 51, 192
KlgPfFrameTab() 57, 193
KlgPfGlobalGet() 7
KlgPfImageAlignment 305
KlgPfImageGetVisibleBoundingBox() 197
KlgPfImageSetVisibleBoundingBox() 198
KlgPfImageType 305
KlgPfIOFilename 113, 223
KlgPfIOStream 113, 223
KlgPfIOTypeRec 305
KlgPfLayoutStatus 305
KlgPfListAdd() 120, 202
KlgPfListAppend() 120, 203
KlgPfListCreate() 119, 203
KlgPfListDelete() 121, 203
KlgPfListDeleteAll() 204
KlgPfListDestroy() 119, 204
KlgPfListDestroyObject() 121, 204
KlgPfListDestroyXtPointer() 121, 204
KlgPfListDuplicate() 119, 205
KlgPfListFind() 122, 205
KlgPfListGetChangedProc() 205
KlgPfListGetCompareProc() 122, 206
KlgPfListGetItem() 123, 206
KlgPfListGetItemCount() 124, 206
KlgPfListGetItems() 124, 206
KlgPfListGetItemSize() 124, 206
KlgPfListGetUserData() 206
KlgPfListInsertAfter() 207
KlgPfListInsertBefore() 120, 207
KlgPfListIsList() 122, 207
KlgPfListIsSorted() 208
KlgPfListItemCompareProc 122
KlgPfListLookupItem() 122, 208
KlgPfListMoveItemsAfter() 208
KlgPfListMoveItemsBefore() 125, 209
KlgPfListRemove() 120, 209
KlgPfListRemoveAll() 120, 209
KlgPfListSearchForward() 124, 210
KlgPfListSearchReverse() 210
KlgPfListSetChangedProc() 126, 210
KlgPfListSetCompareProc() 122, 124, 211
KlgPfListSetDestroyProc() 121, 211
KlgPfListSetItem() 123, 212
KlgPfListSetUserData() 125, 212
KlgPfListSort() 124, 212
KlgPfObjectGetClass() 301
KlgPfObjectGetParent() 301
KlgPfObjectIsColor() 141
KlgPfObjectIsDisplayX11() 145
KlgPfObjectIsDoc() 161
KlgPfObjectIsFont() 169
KlgPfObjectIsFontFamily() 174
KlgPfObjectIsFrame() 193
KlgPfObjectIsPage() 215
KlgPfObjectIsPrinter() 224
KlgPfObjectIsPtrOption() 238
KlgPfObjectIsPtrOptVal() 240
KlgPfObjectIsStyle() 245
KlgPfObjectIsTab() 256
KlgPfObjectIsTable() 268
KlgPfObjectIsTableCell() 269
KlgPfObjectIsTableColumn() 269
KlgPfObjectIsTableRow() 269
KlgPfOutputCallbackStruct 114, 305
KlgPfOutputPolicy 306
KlgPfPageCallbackStruct 149, 306
KlgPfPageCopy() 215
KlgPfPageCreate() 43, 215
KlgPfPageDestroy() 216
KlgPfPageGetColor() 216
KlgPfPageGetFlowFrameList() 216
KlgPfPageGetFlowPageTemplate() 216
KlgPfPageGetFlowSectionTemplate() 216
KlgPfPageGetFrameList() 216
KlgPfPageGetHeight() 217
KlgPfPageGetLayoutStatus() 217
KlgPfPageGetName() 217
KlgPfPageGetParent() 217
KlgPfPageGetType() 217
KlgPfPageGetWidth() 217
KlgPfPageGetX() 217
KlgPfPageGetY() 218
KlgPfPageSetColor() 218
KlgPfPageSetFlowPageTemplate() 218
KlgPfPageSetFlowSectionTemplate() 218
KlgPfPageSetHeight() 218
KlgPfPageSetLayoutStatus() 219
KlgPfPageSetWidth() 219
KlgPfPageSetX() 219
KlgPfPageSetY() 219
KlgPfPageStringToFrame() 57, 219
KlgPfPageTemplateCreate() 32, 38, 43, 46, 220
KlgPfPageType 306
KlgPfParagraphAlignment 306
KlgPfPoint 62
KlgPfPrinterClearSelectedOptions() 224
KlgPfPrinterCreatePCL() 224
KlgPfPrinterCreatePS() 112, 114, 225
KlgPfPrinterCreateX11() 116, 225
KlgPfPrinterDestroy() 12, 225
KlgPfPrinterGetCollate() 225
KlgPfPrinterGetCurrentDocument() 225
KlgPfPrinterGetDisplayX11() 226
KlgPfPrinterGetFilename() 226
KlgPfPrinterGetFontFamilyList() 226
KlgPfPrinterGetFontList() 226
KlgPfPrinterGetName() 226
KlgPfPrinterGetNumCopies() 226
KlgPfPrinterGetOptionValuesByOptionName() 227
KlgPfPrinterGetOutputPageEnd() 227
KlgPfPrinterGetOutputPageStart() 227
KlgPfPrinterGetOutputType() 227
KlgPfPrinterGetPrinterOptionsList() 227
KlgPfPrinterGetPrinterType() 227
KlgPfPrinterGetReverseOrder() 228
KlgPfPrinterGetSelectedOptionsList() 228
KlgPfPrinterGetStream() 228

KlgPfPrinterGetType() 228
KlgPfPrinterOutput() 11, 229
KlgPfPrinterSetCollate() 110, 229
KlgPfPrinterSetDisplayX11() 229
KlgPfPrinterSetFilename() 113, 114, 229
KlgPfPrinterSetNumCopies() 110, 229
KlgPfPrinterSetOutputCallback() 115, 230
KlgPfPrinterSetOutputPageEnd() 111, 230
KlgPfPrinterSetOutputPageStart() 111, 230
KlgPfPrinterSetOutputType() 113, 114, 230
KlgPfPrinterSetPrinterOption() 111, 231
KlgPfPrinterSetReverseOrder() 111, 231
KlgPfPrinterSetStream() 231
KlgPfPrinterStringToFontFamily() 72, 231
KlgPfPrinterTypeEnum 306
KlgPfPtrOptionCreate() 238
KlgPfPtrOptionDestroy() 238
KlgPfPtrOptionFindOptionByName() 238
KlgPfPtrOptionGetDefaultValue() 239
KlgPfPtrOptionGetDescription() 239
KlgPfPtrOptionGetName() 239
KlgPfPtrOptionGetOptionValuesByName() 239
KlgPfPtrOptionGetValuesList() 239
KlgPfPtrOptionSetDefaultValue() 240
KlgPfPtrOptValCreate() 240
KlgPfPtrOptValDestroy() 241
KlgPfPtrOptValGetDescription() 241
KlgPfPtrOptValGetName() 241
KlgPfPtrOptValGetOptionName() 241
KlgPfRectangle 64, 307
KlgPfSectionCallbackStruct 150, 307
KlgPfStyleClearTabs() 246
KlgPfStyleCopy() 68, 69, 246
KlgPfStyleCreate() 68, 246
KlgPfStyleDestroy() 246
KlgPfStyleGetAlignment() 247
KlgPfStyleGetColor() 247
KlgPfStyleGetFont() 247
KlgPfStyleGetFontFamily() 247
KlgPfStyleGetFontSize() 247
KlgPfStyleGetFontStyle() 247
KlgPfStyleGetIndentFirstLine() 248
KlgPfStyleGetIndentLeft() 248
KlgPfStyleGetIndentRight() 248
KlgPfStyleGetLineSpacing() 248
KlgPfStyleGetName() 248
KlgPfStyleGetParagraphSpacing() 248
KlgPfStyleGetSubscriptRatio() 249
KlgPfStyleGetTabList() 249
KlgPfStyleGetTextPosition() 249
KlgPfStyleGetUnderline() 249
KlgPfStyleSetAlignment() 75, 249
KlgPfStyleSetColor() 249
KlgPfStyleSetFont() 250
KlgPfStyleSetFontFamily() 72, 250
KlgPfStyleSetFontSize() 74, 250
KlgPfStyleSetFontStyle() 70, 74, 250
KlgPfStyleSetIndentFirstLine() 69, 76, 250
KlgPfStyleSetIndentLeft() 76, 251
KlgPfStyleSetIndentRight() 76, 251
KlgPfStyleSetLineSpacing() 76, 251
KlgPfStyleSetParagraphSpacing() 77, 251
KlgPfStyleSetSubscriptRatio() 74, 251
KlgPfStyleSetTabsByValueList() 81, 252
KlgPfStyleSetTabsEquallySpaced() 81, 252
KlgPfStyleSetTextPosition() 74, 252
KlgPfStyleSetUnderline() 252
KlgPfTabCopy() 256
KlgPfTabCreate() 15, 79, 256
KlgPfTabDestroy() 257
KlgPfTabGetFillType() 257
KlgPfTabGetTabType() 257
KlgPfTabGetX() 257
KlgPfTableAddHeaderRow() 269
KlgPfTableAddRange() 93, 269
KlgPfTableAddRow() 93, 270
KlgPfTableBorderCopy() 270
KlgPfTableBorderCreate() 270
KlgPfTableBorderDestroy() 270
KlgPfTableBorderGetColor() 271
KlgPfTableBorderGetRuleType() 271
KlgPfTableBorderGetSpacing() 271
KlgPfTableBorderGetThickness() 271
KlgPfTableBorderGetWidth() 271
KlgPfTableBorderSetColor() 271
KlgPfTableBorderSetSpacing() 272
KlgPfTableCellGetAlignment() 272
KlgPfTableCellGetBackgroundColor() 273
KlgPfTableCellGetBorder() 273
KlgPfTableCellGetFrame() 273
KlgPfTableCellGetHeight() 273
KlgPfTableCellGetMarginBottom() 273
KlgPfTableCellGetMarginLeft() 274
KlgPfTableCellGetMarginRight() 274
KlgPfTableCellGetMarginTop() 274
KlgPfTableCellGetSpanColumns() 274
KlgPfTableCellGetSpanOrigin() 274
KlgPfTableCellGetSpanRows() 274
KlgPfTableCellGetStyle() 275
KlgPfTableCellGetWidth() 275
KlgPfTableCellIsSpanned() 275
KlgPfTableCellSetAlignment() 275
KlgPfTableCellSetBackgroundColor() 275
KlgPfTableCellSetBorder() 276
KlgPfTableCellSetHeight() 99, 276
KlgPfTableCellSetMarginBottom() 276
KlgPfTableCellSetMarginLeft() 276
KlgPfTableCellSetMarginRight 277
KlgPfTableCellSetMarginRight() 277
KlgPfTableCellSetMarginTop() 277
KlgPfTableCellSetStyle() 277
KlgPfTableColumnGetAlignment() 277
KlgPfTableColumnGetBackgroundColor() 278
KlgPfTableColumnGetBorder() 278
KlgPfTableColumnGetCellMarginBottom() 278
KlgPfTableColumnGetCellMarginLeft() 278
KlgPfTableColumnGetCellMarginRight() 278
KlgPfTableColumnGetCellStyle() 279
KlgPfTableColumnGetWidth() 279
KlgPfTableColumnSetAlignment() 279

KlgPfTableColumnSetBackgroundColor() 279
KlgPfTableColumnSetBorder() 279
KlgPfTableColumnSetCellMarginBottom() 280
KlgPfTableColumnSetCellMarginLeft() 280
KlgPfTableColumnSetCellMarginRight() 280
KlgPfTableColumnSetCellMarginTop() 280
KlgPfTableColumnSetCellStyle() 281
KlgPfTableCopy() 281
KlgPfTableCreate() 281
KlgPfTableCreateSimple() 94, 282
KlgPfTableDestroy() 282
KlgPfTableDirectionPriority 307
KlgPfTableGetAlignment() 282
KlgPfTableGetBackgroundColor() 282
KlgPfTableGetBorder() 283
KlgPfTableGetBorderMode() 283
KlgPfTableGetCell() 283
KlgPfTableGetCellAlignment() 284
KlgPfTableGetCellFrame() 284
KlgPfTableGetCellMarginLeft() 285
KlgPfTableGetCellMarginRight() 285
KlgPfTableGetCellMarginTop() 285
KlgPfTableGetCells() 284
KlgPfTableGetCellStyle() 285
KlgPfTableGetColumn() 284
KlgPfTableGetColumnList() 285
KlgPfTableGetHeaderTable() 95, 286
KlgPfTableGetOverflowMode() 286
KlgPfTableGetPriorityDirection() 286
KlgPfTableGetRow() 286
KlgPfTableGetRowList() 286
KlgPfTableOverflowMode 307
KlgPfTablePrintAndDestroy() 287
KlgPfTablePrintToCell() 93, 287
KlgPfTableRowGetAlignment() 287
KlgPfTableRowGetBackgroundColor() 287
KlgPfTableRowGetBorder() 288
KlgPfTableRowGetCellList() 288
KlgPfTableRowGetCellMarginBottom() 288
KlgPfTableRowGetCellMarginLeft() 288
KlgPfTableRowGetCellMarginRight() 288
KlgPfTableRowGetCellMarginTop() 288
KlgPfTableRowGetCellStyle() 288
KlgPfTableRowSetAlignment() 289
KlgPfTableRowSetBackgroundColor() 289
KlgPfTableRowSetBorder() 289
KlgPfTableRowSetCellMarginBottom() 289
KlgPfTableRowSetCellMarginLeft() 290
KlgPfTableRowSetCellMarginRight() 290
KlgPfTableRowSetCellMarginTop() 290
KlgPfTableRowSetCellStyle() 290
KlgPfTableRuleType 307
KlgPfTableSetAlignment() 96, 291
KlgPfTableSetBackgroundColor() 97, 291
KlgPfTableSetBorder() 100, 292
KlgPfTableSetBorderMode() 291
KlgPfTableSetCellAlignment() 98, 293
KlgPfTableSetCellMarginBottom() 99, 293
KlgPfTableSetCellMarginLeft() 99, 293
KlgPfTableSetCellMarginRight() 99, 293
KlgPfTableSetCellMarginTop() 99, 294
KlgPfTableSetCellStyle() 97, 294
KlgPfTableSetOverflowMode() 100, 294
KlgPfTableSetPriorityDirection() 107, 294
KlgPfTableSpanCells() 295
KlgPfTabSetFillType() 80, 257
KlgPfTabSetTabType() 80, 258
KlgPfTabSetX() 258
KlgPfTabType 308
KlgPfTextPosition 308
KlgPfUnit 308
KlgPfXmPreviewCreate() 116, 235
KlgPfXmPreviewGetDocument() 235
KlgPfXmPreviewGetPageNumber() 235
KlgPfXmPreviewGetPrinter() 236
KlgPfXmPreviewPopdown() 116, 236
KlgPfXmPreviewPopup() 116, 236
KlgPfXmPreviewSetDocument() 117, 236
KlgPfXmPreviewSetPageNumber() 117, 236
KlgPfXmPreviewUpdate() 116, 236
KlgPfXmPrintDialogCreate() 117, 233
KlgPfXmPrintDialogPopdown() 117, 234
KlgPfXmPrintDialogPopup() 117, 234
KlgPfXmPrintDialogUpdate() 117, 234
klgpr_auth Program
  Use of 8
KLGPRHOME Environment Variable 7

L

LayoutStatus 114, 176, 214
Learning PageFormatter 15
Left Indent 76
Line Spacing 76
LineSpacing 76, 244
LineWidth 62
List Attributes 21
List Object 119
Lists
  Adding List Items 120
  Callback Structure 126
  Checking a List 122
  Creating 119
  Deleting List Items 120
  Destroying 119
  Duplicating 119
  Finding an Item in a List 122
  Relocating List Elements 125
  Retrieving an Item By Index 123
  Retrieving List Information 124
  Retrieving the List Array 124
  Searching and Retrieving List Items 121
  Searching for an Item 124
  Sorting a List 124
  User-Defined Data 125
Location 176, 214

M

Macros 297, 301
  Page Number 59
MarginBottom 53, 99, 176, 261
MarginLeft 53, 99, 177, 261
MarginRight 53, 99, 177, 261
Margins 99
MarginTop 53, 99, 177, 261
MemoryMax 148
Method Errors 26
Multiple Copies 110
Multiple Page Templates 44

N

Name 148, 237
NumCopies 222

O

Option Object 237
OutputCallback 114, 222
OutputPageEnd 222
OutputPageStart 222
OutputPolicy 149
OutputType 113, 114, 223
Overflow 99
OverflowMode 100

P

Page Color, Setting in Template Builder 128
Page Flow, Setting in Template Builder 131
Page Number Macros 301
Page Numbers 59
Page Object 19, 37, 213
Page Order, Setting in Template Builder 129
Page Size, Setting in Template Builder 129
Page Unit 40
PageCallback 33, 34, 149
PageCount 29, 149
PageFormatter Announcements 4
PageList 149
PageNumber 29, 149
PageTemplate 39
PageTemplateList 29, 38, 43, 149
PaperSizeUnits 223
Paragraph Appearance
  Alignment 75
  Changing 75
  Indents 76
  Line Spacing 76
  Spacing 77
ParagraphSpacing 77, 245
Parser Demo 310
PCL Printer 112
PDF 113
PostScript Printer 112
PPD File 112
Print Dialog 233
Print Dialog Object 117
Print Previewer 115
Printer 150, 235
Printer Attribute 28
Printer Object 19, 109, 221
Printer Options 111
PrinterOptionList 223
PrinterOptionSelectedList 111, 223
PrinterOptionsList 111
printers file 112
PrinterType 223
Printing 24
  Collating 110
  File 114
  Multiple Copies 110
  Output Policy 114
  Output Type 113, 114
  Range 111
  Reverse Order 111
  Stream 113
  To A File 114
PtrOption Object 237
PtrOption object 111
PtrOptionValue Object 237
PtrOptionValue object 111

R

Realestate demo 310
Reason 33, 126
Registering Procedures 33, 115, 126
Return Types, On Error 26
Reverse Order Printing 111
ReverseOrder 111, 223
Right Indent 76
Row Height 99
RowList 268
RuleType 260
Running the Parser 311

S

Section Break, Adding in Template Builder 131
Section Breaks 32
SectionCallback 33, 34, 150
Set Method 21
Setting Page Attributes 44
Setting Printer Attributes 110
Setting Style Attributes 68
Sorted 202
Spacing 260
SpanColumns 262
Spanned 262
SpanOrigin 262

SpanRows 262
Stream 223
Style
  Copying 68
Style Object 67, 243
Style object 30
StyleList 68, 150
StyleList Attribute 57
StyleList list 30
SubscriptRatio 74, 245
Subscripts/Superscripts 74
SupportedSizes 168

T

Tab List 77
Tab Object 15, 79, 255, 259
Table Inheritance 104
TableList 150
TabList 79, 245
Tabs
  Fill Type 80
  Flow Tab 15, 81
  Type 80
TabType 256
Template Builder, Adding a Frame in 129
Template Builder, Setting Frame Color 129
Template Builder, Setting Frame Size in 130
Template Builder, Setting Page Color 128
Template Builder, Setting Page Order 129
Template Builder, Setting Page Size 129
Templates, Sample 128
text flow, setting in Template Builder 131
TextPosition 74, 245
TIFF 85
Type 214, 223
Typographical Conventions 1

U

Underline 245
Underlining 75
Unit 150
Unit Attribute 51
Unit Conversion Macros 297
Units 22
User Data 25
UserData 202
Using Template Builder 127

V

ValueList 111, 238

W

Webprint Demo 310
Width 39, 235, 260, 263
Window 143
Writing to a Template Frame 55

X

Xparser 310