Content Manager OnDemand

AFP2PDF Transform User's Guide

IBM®

SC19-2944-00

Content Manager OnDemand

AFP2PDF Transform User's Guide

IBM®

SC19-2944-00

This edition replaces SC19-1287-01.

Contents

Figures

Tables

About this publication

This publication provides information about using AFP2PDF Transform. This publication helps you:

- Plan for transforming data from Advanced Function Presentation (AFP) format to Adobe® Acrobat Portable Document Format (PDF).
- Install and configure AFP2PDF Transform.
- Map fonts and images.
- Use application programming interfaces (APIs).

The information in this publication is for system programmers who install and configure AFP2PDF Transform, and for operators who use AFP2PDF Transform. This publication assumes that you are experienced using Microsoft® Windows®, UNIX®, or IBM® i systems, or the OnDemand or eClient applications of IBM DB2® Content Manager.

Understanding syntax notation

These rules apply to syntax and coding illustrations throughout this publication:

- Bold highlighting identifies commands and other items whose names are predefined by the system, information you should actually type, or the actual value you should set, such as **True**.
- Variable data is printed in italics. Enter specific data to replace the characters in italics; for example, for *filename* you could enter **Data.afp**. Italics also identify the names of publications.
- Monospacing identifies examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or files and directories; for example, `alias.fnt`.
- Do not enter these symbols as part of a parameter or option:
  Vertical Bar |
  Underscore ___
  Brackets [ ]
  Braces { }
  Ellipsis ...
- A vertical bar between two values means that you select only one of the values.
- An underscored value means that if an option is not specified, the underscored value, called the default, is used.
- Brackets around a value mean that you do not have to select the value; the value is optional.
- Braces around a value mean that you must select one of the mutually exclusive values. For example, **{ THIS | THAT }**
- An ellipsis following a command or set of commands indicates the command or set of commands can be repeated.

Related information

For information about the AFP2HTML Transform, see *Data Transforms AFP2HTML Transform User's Guide*, SC19-1288.

For information about AFP, see this Web site: www.infoprint.com

Summary of Changes

Summary of Changes for *AFP2PDF Transform User's Guide*, SC19–1287–02

This publication contains additions and changes to information previously presented in *AFP2PDF Transform User's Guide*, SC19–1287–01. The technical additions and changes are marked with revision bars (|) in the left margin.

These changes have been made throughout the publication:

- References to the i5/OS® operating system have been changed to "IBM i".
- A note has been added indicating that the file examples are specified for the Windows environment but to use the examples in a UNIX environment, use the UNIX file naming convention for any file name. For example, the file `c:\images\form1.jpg` in Windows could be `/tmp/form1.jpg` in a UNIX environment.

This information is new or updated:

- "AFP2PDF Transform limitations" on page 1 has been updated.
- IBM i Common Server has been added to the server requirements in Chapter 2, "Installing AFP2PDF Transform," on page 3.
- IBM i Portable Application Solutions Environment (PASE) has been added to the UNIX environments in "Installing AFP2PDF Transform on AIX, HP-UX, Solaris, and Linux" on page 3.
- A new section, "Installing AFP2PDF Transform on an IBM i OnDemand server" on page 3, has been added.
- The documentation for "Installing AFP2PDF Transform with OnDemand" on page 4 has been updated.
- A new section, "Installing AFP2PDF Transform with eClient" on page 4, has been added for using the AFP2PDF Transform with the IBM DB2 Content Manager eClient.
- In the AFP2PDF Transform options file, these parameters have been added:
  - DotDensity on page 13
  - GOCA_Pattern on page 14
  - Ignore_Data_Font_Height on page 15
  - TrueType_Directory on page 18
  - UDC_Range on page 18

  Also, the Locale_Path parameter on page 15 has been updated.
- The **split_afp2pdf** command has been updated with the -k parameter on page 21.
- Table 1 on page 25 for the AFP2PDF Transform font support files has been updated with TrueType outline fonts files.
- The "wincp" attribute in Table 5 on page 29 for CODEPG attributes has been updated.
- A new section, "Mapping AFP TrueType fonts" on page 37, has been added.
- The types of AFP2PDF Transform APIs have been updated in Chapter 6, "Application programming interfaces," on page 47.

- A new section that describes API packaging information has been added for “IBM i OnDemand server” on page 48.
- A new section that describes dynamically loading the API code has been added for “IBM i OnDemand server” on page 52.
- The definition of "integrated file system" has been added to the “Glossary” on page 65.

Chapter 1. Overview

AFP2PDF Transform converts Advanced Function Presentation (AFP) documents into Adobe Acrobat Portable Document Format (PDF) files. AFP2PDF Transform maps AFP format to PDF format exactly, making it a more robust solution than AFP2HTML.

AFP2PDF Transform lets you:

- Operate on multiple operating systems, including AIX®, HP-UX, IBM i, Linux®, Sun Solaris, Windows, and z/OS®.
- Integrate multiple servers with little to no client workstation modifications. AFP2PDF Transform runs on your Web server or other back-end application server.
- View documents with the same fidelity as if they were printed. If the Adobe Acrobat plug-in is installed with a Web browser, you can view and print these documents within the browser application.
- Use configuration files to customize how AFP documents are transformed.
- Fully integrate with the IBM DB2 Content Manager OnDemand Web Enablement Kit and the IBM Enterprise Information Portal (EIP).
- Fully integrate with the IBM DB2 Content Manager eClient.

Benefits

AFP2PDF Transform gives you these added benefits:

- Reduce costs associated with printing and mailing by delivering documents electronically.
- Quickly retrieve your information within multi-page documents using Adobe Acrobat search and navigation features.
- Print AFP2PDF Transform documents on any local printer using the print function in Adobe Acrobat.
- Increase control over your information with an added layer of security and encryption. Define an owner to control modification, copying and printing; define end-user passwords to control document access; and add a digital signature to provide more security.

AFP2PDF Transform limitations

The current limitations with the AFP2PDF Transform include:

- The following set of bar codes are generated in the output PDF when using the Bar Code Object Content Architecture (BCOCA) in AFP data. Any other bar codes are ignored during conversion.
  - Code 39 (3-of-9 Code), AIM USS-39
  - Interleaved 2-of-5, AIM USS-I 2/5
  - POSTNET
  - Code 128, AIM USS-128
  - Japan Postal Bar Code
  - Data Matrix
  - USPS Four-State

  - PDF417
  - QR code
  - Codabar
- When AFP data formatted for N-up partitioning is encountered, the multiple partitions that make up the physical AFP page are converted to separate pages in the PDF file.
- Only limited support is available for object containers with TIFF and JFIF image formats.

Chapter 2. Installing AFP2PDF Transform

These are the server requirements and the client requirements for AFP2PDF Transform:

- Server requirements
  - HP-UX 11.0 for Itanium or later
  - IBM AIX 5.1 or later
  - IBM i Common Server 5.4 or later
  - IBM z/OS UNIX System Services V1.8 or later
  - Microsoft Windows 2003 Server R2 or later
  - Linux Kernel 2.4.5 or later (IBM System x/System p/System z)
  - Sun Solaris 8 or later (SPARC only)
- Client requirements
  - Adobe Acrobat, Acrobat Reader, or Acrobat Plug-In 5.0 or later (Digital certificate support requires 7.0.5 or later)

If you plan to use the transform in conjunction with the IBM DB2 Content Manager OnDemand Web Enablement Kit, install the AFP2PDF Transform on the same workstation or server.

To use the transform with the IBM DB2 Content Manager eClient, install the AFP2PDF Transform on the eClient server.

Installing AFP2PDF Transform on Windows

To install AFP2PDF Transform on Windows, run the `afp2web.exe` file. By default, all files are installed to the `C:\Program Files\IBM\AFP2web` directory.

Installing AFP2PDF Transform on AIX, HP-UX, Solaris, and Linux

To install AFP2PDF Transform on AIX, HP-UX, Solaris, and Linux, run the `afp2web` file. By default, all files are installed to the *Content Manager OnDemand server directory*/`afp2web` directory.

Installing AFP2PDF Transform on an IBM i OnDemand server

For an IBM i server with Content Manager OnDemand, the AFP2PDF Transform is delivered in two save files, QRLMINSA2P (installation code) and QRLMA2P (installation objects).

To install the transform:

1. Copy both save files to the QRDARS library on your IBM i system.
2. Enter this command to restore the installation program from the QRLMINSA2P save file:
   ```
   RSTOBJ OBJ(QRLMINSA2P) SAVLIB(QRDARS) DEV(*SAVF) SAVF(QRDARS/QRLMINSA2P) RSTLIB(QRDARS)
   ```
3. Run this program call:
   ```
   CALL QRDARS/QRLMINSA2P
   ```

Notes:

1. Symbolic links to the directory object in `/QIBM/ProdData/OnDemand/www/binpdf` have been created in `/QIBM/UserData/OnDemand/www/binpdf`.
2. IBM recommends that you use the `UserData` directory when referencing the installed objects. If you create any additional objects, do not save them in the `/QIBM/ProdData/OnDemand/www/binpdf` directory, which might get replaced when upgrading to a new release or installing PTF updates.

Installing AFP2PDF Transform with OnDemand

To use the AFP2PDF Transform with the Content Manager OnDemand Web Enablement Kit, see the appropriate documentation for more details about configuring the programs to operate together:

- *IBM DB2 Content Manager OnDemand for Multiplatforms: Web Enablement Kit Implementation Guide*, SC18-9231
- *IBM DB2 Content Manager OnDemand for z/OS and OS/390: Web Enablement Kit Implementation Guide*, SC18-1215
- *IBM Content Manager OnDemand for i: Common Server Web Enablement Kit Installation and Configuration Guide*, SC27-1163 (Versions 5.4 and 6.1) or SC19–2791 (Version 7.1 or later)

Specifically, you must modify the `arswww.ini` file to call the AFP2PDF Transform when processing an AFP file. At a minimum, you must make these configuration changes:

- The AFPViewing option must be “pdf” in the browser sections.
- The InstallDir option in the afp2pdf section must point to the directory on the server that contains the AFP2PDF Transform.

Installing AFP2PDF Transform with eClient

To use the AFP2PDF Transform with the Content Manager eClient, the AFP2PDF Transform must be installed on the same server where the eClient is installed. Refer to *IBM DB2 Content Manager: Installing, Configuring, and Managing the eClient*, SC27–1350, to install the eClient. See “Installing AFP2PDF Transform on Windows” on page 3 or “Installing AFP2PDF Transform on AIX, HP-UX, Solaris, and Linux” on page 3 to install the AFP2PDF Transform. Be sure to make note of the directory where you installed the transform.

To use the AFP2PDF Transform with the Content Manager eClient:

1. Locate the eClient configuration file, `IDMadminDefaults.properties`:
   - For eClient 8.3, the default location is:
     - `%IBMCMROOT%\CMeClient` on Windows
     - `$IBMCMROOT/CMeClient` on UNIX systems

     where IBMCMROOT is the location of the IBM directory for the eClient program files. For example, `c:\program files\IBM` on Windows.
   - For eClient 8.4, the default location is:
     - `*was_profile_home*\installedAPPs\*hostname*Node01Cell\eClient.ear\eclient.war` on Windows
     - `*was_profile_home*/installedAPPs/*hostname*Node01Cell/eClient.ear/eclient.war` on UNIX systems

     where *was_profile_home* is the location of the WebSphere application server profile for the server that runs the eClient WebSphere application. For example, c:\program files\IBM\WebSphere\AppServer\profiles\AppSrv01 on Windows or /opt/IBM/WebSphere/AppServer/profiles/AppSrv01 on UNIX.

2. Edit IDMadminDefaults.properties and replace the application/pdf=launch line with **application/pdf=don't launch**. Setting the application/pdf type to "don't launch" indicates that the PDF application should be converted to a viewable format that can be handled by a browser.
3. From the AFP2PDF Transform directory, ...\java_api on Windows or .../java_api on UNIX systems, run this command:
   ```
   java -jar a2peip.jar AFP2PDF
   ```
4. Follow the prompts and specify this information:
   - The fully qualified path where the AFP2PDF Transform is installed.
   - The fully qualified path where the cmbview81.jar file is installed. The default location is %IBMCMROOT%\lib on Windows or $IBMCMROOT/lib on UNIX systems.
   - The fully qualified path where the a2peip.jar file is installed.
   - **TRUE** to enable logging or **FALSE** to turn logging off.
5. Correct any errors that are displayed on the console. Otherwise, a new jar file named cmbview81.jar.NEW is created in the same directory as the original.
6. Rename the original cmbview81.jar file to **cmbview81.jar.original** in case you need to recover your system later.
7. Rename cmbview81.jar.NEW to **cmbview81.jar**.
8. Make sure the new cmbview81.jar file has the same permissions and ownership as cmbview81.jar.original.
9. Restart the eClient server.

You can now use AFP2PDF Transform with the eClient.

Removing AFP2PDF Transform

On Windows, to remove AFP2PDF Transform, select **Start -> Control Panel -> Add or Remove Programs**.

On AIX, HP-UX, Solaris, and Linux, to remove AFP2PDF Transform, run the *Content Manager OnDemand server directory*/afp2web/_uninst850afp2web/uninstallafp2web file.

Chapter 3. Configuration

This chapter describes the commands and files used during the configuration process. It also describes AFP resources and security for the transform.

Note: The directory and file examples in this chapter are specified for the Windows environment. To use these examples in a UNIX environment, use the UNIX file naming convention for any file name. For example, `\font\maps` in Windows is `/font/maps` in a UNIX environment, or the input AFP file `c:\documents\afpdoc.afp` in Windows would be `/documents/afpdoc.afp` in a UNIX environment.

AFP2PDF Transform command

The **afp2pdf** command transforms AFP files and resources into PDF files that you can distribute over the Internet and view with a Web browser installed with the Adobe Acrobat plug-in. If the plug-in is not installed on your Web browser, you can view the PDF files with Adobe Acrobat Reader.

Syntax

Only a portion of the parameters needed to control the AFP2PDF Transform are available with the command. The other parameters are specified in an options file. By default, this file is named `a2pxopts.cfg` and must reside in the same directory as the program module. See "AFP2PDF Transform options file" on page 10 for more information about the options file.

The syntax for the **afp2pdf** command is:

```
►►──afp2pdf──┬──────────┬──┬───────┬──┬────────┬──────────────────────────────►
             └─-a──codes─┘  └─-e──pw─┘  └─-f──def─┘

►──┬──────────────────────────┬──┬────────────┬──┬────┬──┬────┬────────────────►
   └─-g──(YYYYMMDDHHmmSSOHH'mm')─┘  └─-i──optfile─┘  └─-l─┘  └─-m─┘

►──┬─────────────┬──┬────────────┬──┬────────────┬──┬────────────┬───────────►
   └─-n──fontpath─┘  └─-o──outfile─┘  └─-p──page_no─┘  └─-r──resfile─┘

►──┬──────────────────────────┬──┬─────────┬──┬───────┬──┬────────────┬──────►
   └─-s──(YYYYMMDDHHmmSSOHH'mm')─┘  └─-t──turn─┘  └─-u──pw─┘  └─-v──mapfile─┘

►──afpfile─────────────────────────────────────────────────────────────────►◄
```

Parameters

-a *codes*
When used with an owner password (`-e` *pw*), specifies which PDF display functions are restricted. The codes for the display functions, which can be used in any order, are:

a	Add or modify text annotations and interactive form fields.
c	Modify the document contents.
p	Print the document.

s Copy text and graphics from the document.

-e *pw*
Specifies an alphanumeric password that gives permission to change document security settings (also known as a *master* or *owner* password). For example, if the printing function in the PDF display has been restricted (`-a p`), you must supply a password to override the setting.

-f *def*
Specifies the fully qualified file name of the form definition resource that is used when transforming an AFP file. For example:

```
afp2pdf -f c:\mydirectory\myformdef.fde afpfile.afp
```

-g (*YYYYMMDDHHmmSSOHH'mm'***)**
Sets a time stamp for when the document was created, where:

- **YYYY** is the year
- **MM** is the month
- **DD** is the day (01-31)
- **HH** is the hour (00-23)
- **mm** is the minute (00-59)
- **SS** is the second (00-59)
- **O** is the relationship of local time to Universal Time (UT), denoted by one of the characters +, -, or Z
- **HH** is the absolute value of the offset from UT in hours (00-23)
- **mm** is the absolute value of the offset from UT in minutes (00-59)

-i *optfile*
Specifies the location and name of the transform options file that is different from the default. This file name should not use relative paths and should be fully qualified. See "AFP2PDF Transform options file" on page 10 for more information about the options file.

-l Turns off all of the generated error and informational console messages and sends them to the `afp2pdf.err` log file.

-m Generates a linearized PDF file which reorganizes the data for more efficient processing in a network environment. Also known as Fast Web View PDF documents, files generated with this parameter make it possible to view very large documents without long download delays. To use this parameter, verify that the Web server or Web application sending this PDF data over the network provides the page-by-page downloading or "byte-serving" function.

-n *fontpath*
Specifies the location of the font definition files needed by the transform. This parameter should be used if the font files are not located in the `\font` subdirectory where the transform modules were installed.

-o *outfile*
Specifies the location and file name of the output PDF file that is different from the default output file, which has the same location and name as the input file, but with a file extension of ".pdf". For example, when the PDF is generated from an AFP file named `afpdoc.afp`, a file named `afpdoc.pdf` is created. You can use this parameter to put the output file in a different directory than the input file or give it a different file name.

-p *page_no*
Specifies the page number that is to be transformed in the AFP document. If

the specified page number does not fall within the document page range, the first page in the document is transformed.

-r *resfile*
Specifies the fully qualified file name of the AFP resource group file to be used when transforming the AFP file. For example:

```
afp2pdf -r c:\mydirectory\afpresfile.res afpfile.afp
```

-s (*YYYYMMDDHHmmSSOHH'mm'***)**
Sets a time stamp for when the document was signed, where:

- **YYYY** is the year
- **MM** is the month
- **DD** is the day (01-31)
- **HH** is the hour (00-23)
- **mm** is the minute (00-59)
- **SS** is the second (00-59)
- **O** is the relationship of local time to Universal Time (UT), denoted by one of the characters +, -, or Z
- **HH** is the absolute value of the offset from UT in hours (00-23)
- **mm** is the absolute value of the offset from UT in minutes (00-59)

For example, December 23, 1998, at 7:52 PM, U.S. Pacific Standard Time, is represented by the string: `19981223195200-08'00'`

-t *turn*
Specifies the rotation value to use when transforming the file. Valid values are **0** , **90**, **180**, and **270**. Some AFP files might have already been formatted with a rotated orientation; therefore, you must use this parameter to align the text in an upright position.

-u *pw*
Specifies an alphanumeric password that gives permission to open the PDF document (also known as a *user* password).

-v *mapfile*
Specifies the location and name of the image map configuration file that is different from the default, `imagemap.cfg`. This file name should not use relative paths and should be fully qualified. See "Creating the image map configuration file" on page 39 for more information about the image map configuration file.

afpfile
Specifies the AFP input file that is to be transformed to PDF. This parameter is required.

Return codes

When the **afp2pdf** command runs, you see one of these return code values:

0 Successful completion of the transform.

Nonzero
An error has occurred.

AFP2PDF Transform options file

Parameters to control settings for the AFP2PDF Transform are specified in an options file. By default, the name of this file is `a2pxopts.cfg`. When running the transform function with the **afp2pdf** command line program, the `a2pxopts.cfg` file must reside in the same directory as the program module.

If running the transform function from the API, you can locate the options file in any directory on the system and you can use any file name. By setting the szOptionsFile option in the structure passed to the transform function, different options files can be specified for different types of documents.

Parameters in the options file must be specified on separate lines and have the format “parameter=value”. For example:

```
Disable_Compression=True
Auto_Rotate=True
```

Parameters and values are not case-sensitive. Lines starting with a semicolon (;) or a pound (#) character are comments.

The AFP2PDF Transform option parameters are:

Append_Log_to_PDF=True | <u>False</u>
When **Logging=Buffer**, indicates whether the log file is appended to the PDF output file.

Append_PDF_File=*file*,*n*
Indicates where a specified PDF file is appended to the generated PDF file, where:

file Specifies the name and location of the PDF file that is to be appended to the generated PDF file.

n Specifies **0** for the beginning of the file or **1** for the end of the file.

For example, `Append_PDF_File="C:\term.pdf",0` appends the `c:\term.pdf` file to the beginning of the generated PDF file, while `Append_PDF_File="C:\term.pdf",1` appends the `c:\term.pdf` file to the end of the generated PDF file.

Author=*name*
Specifies 1 to 62 characters for the Author name in the Info Dictionary of the output PDF file. This information is displayed to the user if the "document properties" function is selected in Adobe Acrobat. Enclose the text in double quotes if the value contains blanks. For example: `Author="InfoPrint Solutions"`.

Auto_Rotate=True | <u>False</u>
Indicates whether the transform determines the orientation of each page and rotates it so that the text appears right-side up. By default, the AFP document is converted as is, so if the AFP page is formatted in a rotated orientation, the output PDF is also rotated. Setting this parameter to **True** is useful when pages in a document are rotated with different orientations.

If a document rotation setting is also given (an input parameter to rotate the entire document), the Auto_Rotate parameter overrides this setting.

Bilevel_Image_Cnvt=True | <u>False</u>
Indicates whether the output device for the PDF data is changed to one that supports gray-scale images. On some larger bi-level (black and white) images,

horizontal lines might appear by processing the image in smaller bands. Setting this parameter to **True** might remove the horizontal lines at the band boundaries.

Cache_AFP_Overlay=True | False
Indicates whether AFP overlays are saved during transformation. Setting this parameter to **True** can result in a smaller PDF file and a faster conversion.

Note: This does not work on all AFP files.

Cache_Font_Image=True | False
Indicates whether font images are saved during transformation. Setting this parameter to **False** can result in a larger PDF file; however, it might help the rendering performance of the PDF files when using older Acrobat versions (Acrobat 4) for display.

Certificate=*pkcs12_file***,***pkcs12_password***,***cache_directory*
Specifies the PKCS#12 certificate file used to sign the document, where:

pkcs12_file
Specifies the fully qualified name of the PKCS#12 certificate file.

pkcs12_password
Specifies the password needed to read the PKCS#12 file.

cache_directory
Specifies the fully qualified path to a directory that is used for certificate storage. This is an optional value that can be used to improve the performance of the conversion.

Certificate2=*pkcs12_file***,***pwd_directory***,***cache_directory*
Specifies the PKCS#12 certificate file used to sign the document, where:

pkcs12_file
Specifies the fully qualified name of the PKCS#12 certificate file.

pwd_directory
Specifies the fully qualified path to a file that contains the password needed to read the PKCS#12 file.

cache_directory
Specifies the fully qualified path to a directory that is used for certificate storage. This is an optional value that can be used to improve the performance of the conversion.

Color=*afpname***,***red***,***green***,***blue*
Specifies the RGB value color setting that overrides the display of named color values in the AFP OCA (graphics objects), where:

afpname
Specifies one of these colors defined in AFP: BLUE, BROWN, CYAN, DARKBLUE, DARKCYAN, DARKGREEN, DARKGREY, DARKYELLOW, GREEN, GREY, HIGHLIGHT0, HIGHLIGHT1, HIGHLIGHT2, HIGHLIGHT3, MAGENTA, MUSTARD, ORANGE, PURPLE, RED, WHITE, YELLOW.

red
green
blue
Specifies the RGB value settings in the range of 0 to 255.

For example, the AFP named color BLUE is changed to a light gray color with these values:

```
Color=BLUE,220,220,220
```

Compression_Level=*n*
Specifies the compression level of the Flate compression used in the PDF file, where *n* is 0 to 9. The larger the number the longer the conversion takes, but a smaller PDF file should be created.

Creator=*name*
Specifies 1 to 62 characters for the Creator name in the Info Dictionary of the output PDF file. This information is displayed to the user if the "document properties" function is selected in Adobe Acrobat. Enclose the text in double quotes if the value contains blanks. For example: `Creator="InfoPrint Solutions Company"`.

Default_Encryption_Permissions=[a, c, p, s][[,] 5[, e, d, i, q]]
Sets one or more default encryption permissions to be used when encrypting the PDF output. The permission codes are:

a Adding or changing annotations or form fields is denied.

c Changing the document is denied in Acrobat.

p Printing the document is denied from Acrobat.

s Selecting and copying text and graphics is denied.

5 128-bit encryption is set (supported on Acrobat 5.0 or later; requires a special transform package). You can set one or more of these codes with 128-bit encryption:

- **e** Extracting text and graphics is denied.
- **d** Assembling documents is denied.
- **i** Editing form fields is denied.
- **q** Printing high quality is denied.

Notes:

1. Using any of the 128-bit encryption codes produces a file that can only be opened with Acrobat 5.0 or later.
2. Code 5 can be used in combination with one of the a, c, p, or s codes to force 128-bit encryption without setting the i, e, d, or q codes.

Default_Linearization=True | False
Indicates whether the PDF output is linearized.

Default_Owner_Password=*pw*
Specifies a default owner encryption password, where *pw* is the alphanumeric owner encryption password.

Default_User_Password=*pw*
Specifies a default user encryption password, where *pw* is the alphanumeric user encryption password.

Disable_Bookmark_Generation=True | False
Indicates whether PDF bookmarks are created. If page level indexing information is available in the input AFP document, PDF bookmarks are automatically generated. If you do not want bookmarks created, you must set this parameter to **True** .

Disable_Compression=True | False
Indicates whether the transform compresses PDF data. Setting this parameter to **True** to turn off compression can be useful when trying to determine problems that might exist in the PDF output file.

| **DotDensity,***patternid***,***afpcolorname***=***red***,***green***,***blue*
| Specifies the RGB value color setting for the AFP Graphics Object Content
| Architecture (GOCA) pattern that is used to fill the interior of an area, where:

| *patternid*
| Specifies the GOCA pattern identifier, which defines various dot fill
| patterns of varying density. Pattern ID values of 1 to 8 are supported.

| *afpcolorname*
| Specifies one of these pattern foreground colors defined in AFP: BLACK,
| BLUE, BROWN, CYAN, DARKBLUE, DARKGREEN, DARKTURQUOISE,
| DEFAULT, GREEN, GREY, MAGENTA, MUSTARD, ORANGE, PURPLE,
| RED, WHITE, YELLOW.

| *red*
| *green*
| *blue*
| Specifies the RGB value settings in the range of 0 to 255.

| For example, `DotDensity,5,GREY=220,220,220` specifies the RGB value of
| 220,220,220 for GOCA pattern 5 with a gray foreground color.

Enable_Auto_Font_Image=True | False
Indicates whether a font is mapped automatically. Setting this parameter to **True** causes the font to be converted to image data if the font resources exist and a font mapping does not exist.

Enable_Auto_Image_Cache=True | False
Indicates whether images are saved.

Note: This does not work with all AFP.

Enable_UDC=True | False
Indicates whether user-defined characters are enabled for double-byte languages.

Expand_Index_Values=True | False
Indicates whether the subentries in the bookmark pane are automatically expanded.

FontExt=*.*extension*
Specifies the file extension that is used when searching for font resources in resource directories. For example, if `FontExt=*.240` and the AFP document references a font named `C0FONTCS`, the transform searches for a file named `C0FONTCS.240` in the resource directories.

Note: The *extension* value is case-sensitive on most operating systems.

Font_Image_Pad_Height=*n*
Specifies the number of additional padding rows used when generating raster characters. Larger padding values enhance the display of small characters but increase the PDF output size. The default value is **32**.

Font_Image_Pad_Width=*n*
Specifies the number of additional padding columns used when generating

raster characters. Larger padding values enhance the display of small characters but increase the PDF output size The default value is **32**.

Font_Path=*directory*
Specifies the base directory that the transform uses to search for the font configuration files. By default, this base directory is the `\font` subdirectory where the transform modules were installed. If this parameter is used when the font configuration files have been moved to a different location, the same directory structure of the font configuration files must be preserved. For example, if this entry was given: `Font_Path=c:\fontfils`, the code page map files must reside in the `c:\fontfils\maps` directory.

Global_Scale=*n.n*
Applies a scaling factor to all the data on the page relative to the paper or media represented in the PDF. For example, a scale of 50.0 reduces the data in half relative to the page. The default value is **100.0**.

GOCA_Pass1
Reserved for future use.

GOCA_Pass2
Reserved for future use.

| **GOCA_Pattern=***patternid***,***var1***,***var2*
| Specifies the characteristics for the AFP Graphics Object Content Architecture
| (GOCA) pattern that is used to fill the interior of an area, where:

| *patternid*
| Specifies the GOCA pattern identifier, which defines various line fill
| patterns. Pattern ID values of 9 to 14 are supported.

| *var1*
| Defines the line width.

| *var2*
| Defines the line spacing.

| For example: `GOCA_PATTERN=9,10,250` specifies GOCA pattern 9 with a line
| width of 10 and a line spacing of 250.

GOCA_Use_Circles=True | False
Indicates whether the transform uses circles for some GOCA patterns instead shaded areas. Setting this parameter to **True** creates a large PDF file.

Honor_Constant_Forms=True | <u>False</u>
Indicates whether the transform generates extra pages created from the constant forms control function, which is a function in AFP architecture that produces one or more static overlays on a page without variable data from the print file.

Honor_Media_Eject_Control=True | <u>False</u>
Indicates whether the transform tries to honor all of the media eject controls specified in the AFP form definition resource. Setting this parameter to **True** might cause extra blank pages to be placed in the output PDF.

Honor_Medium_Colored_Rules=True | <u>False</u>
Indicates whether objects that are drawn with the medium color are honored. In the AFP architecture, objects can be drawn with the color of the medium, such as paper. Depending on the order in which objects appear in the AFP data, an object drawn with the color of the medium might cover up an object

that it overlaps. By default, the transform ignores objects that are drawn with the medium color; however, if this parameter is set to **True**, overlapping objects are allowed.

Horizontal_Offset=*n*
Applies a horizontal offset to all the data on the page relative to the paper or media represented in the PDF. The units of this setting are in 1440 units per inch.

Ignore_Data_Font_Height=True | False
Indicates whether the font height is ignored. By default, the transform honors the font height value specified in the AFP data; however, if this parameter is set to **True**, the font height setting in the AFP data is ignored and the font size specified in the font configuration file, `csdef.fnt`, is used.

ImageMapEntries_File=*file*
Specifies the file that the transform uses to output the image information contained in the AFP file. The information in this file can then be modified and passed as input back into the transform function to change how individual AFP images are transformed. See Chapter 5, "Mapping AFP images," on page 39 for more information about mapping images.

Keywords=*text*
Specifies 1 to 120 characters for the Keywords entry in the Info Dictionary of the output PDF file. This information is displayed to the user if the "document properties" function is selected in Adobe Acrobat. Enclose the text in double quotes if the value contains blanks. For example: `Keywords="InfoPrint Solutions Company Advanced Function Presentation"`

Launch_Preview=True | False
Indicates whether the Adobe Acrobat automatically displays the PDF file just created. This parameter is only valid when using the **afp2pdf** command on the Windows server.

Locale_Path=*path*
Specifies the path location of the locale-specific information files needed during the transform process. This parameter should only be used when converting multiple-byte language files (Traditional Chinese, Simplified Chinese, Japanese, Korean, or Unicode).

Logging={True | Append | Buffer},*logdir*
Specifies the type of tracing that is turned on for program debugging and where the log file is located. The values are:

True
Traces on a document boundary and overwrites the trace data from a previous document in the log file.

Append
Appends trace data to the end of the log file.

Buffer
Traces API functions. The A2PGetMessageBuffer API function returns the trace text. When **Append_Log_to_PDF=True**, the log file is appended to the PDF output file.

logdir
Specifies the directory where the log file is saved.

Max_Annotes
Deprecated–supported but no longer used by the transform. The transform automatically adjusts this setting as needed.

Max_Fonts
Deprecated–supported but no longer used by the transform. The transform automatically adjusts this setting as needed.

Max_Images
Deprecated–supported but no longer used by the transform. The transform automatically adjusts this setting as needed.

Max_Leaves
Deprecated–supported but no longer used by the transform. The transform automatically adjusts this setting as needed.

Max_Objects
Deprecated–supported but no longer used by the transform. The transform automatically adjusts this setting as needed.

Max_Overlays
Deprecated–supported but no longer used by the transform. The transform automatically adjusts this setting as needed.

Max_Pages
Deprecated–supported but no longer used by the transform. The transform automatically adjusts this setting as needed.

Modify_Text_Colors=True | False
Indicates whether RGB value color settings override the display of named color values in the AFP text (PTOCA) and AFP graphic objects. See the Color parameter on page 11.

Old_Static_Paper=True | False
Indicates whether the older processing for the Static_Paper_Length and Static_Paper_Width is enabled.

Output_IndexInfo=True | False
Indicates whether the group level index information for the document is placed as a bookmark using the format "index attribute : index value". The bookmark text can then be searched for by other applications processing the PDF file.

Note: Do not set this parameter to **True** on password protected files because the bookmark text becomes encrypted.

OverlayExt=*.*extension*
Specifies the file extension that is used when searching for overlay resources in resource directories. For example, if `OverlayExt=*.OLY` and the AFP document references an overlay named `O1OVERLY`, the transform searches for a file named `O1OVERLY.OLY` in the resource directories.

Note: The *extension* value is case-sensitive on most operating systems.

Page_Rotation=0 | 90 | 180 | 270
Specifies the rotation value to use when transforming the file. Some AFP files might have already been formatted with a rotated orientation; therefore, you must use this parameter to align the text in an upright position.

PageSegExt=*.*extension*
Specifies the file extension that is used when searching for page segment resources in resource directories. For example, if `PageSegExt=*.PSG` and the AFP document references a page segment named `S1PAGSEG`, the transform searches for a file named `S1PAGSEG.PSG` in the resource directories.

Note: The *extension* value is case-sensitive on most operating systems.

PDF/A=True | False
Indicates whether the transform generates output that conforms to the PDF/A-1b specification, which is the format for the long-term preservation of electronic documents. When this parameter is set to **True**, these conditions must be met for the transform to generate output that conforms to the specification:

- All fonts must be either embedded or processed as images for PDF/A-1b compliance. See "Embedding Type 1 fonts" on page 34 for information about embedding fonts; see "Using custom AFP raster font files" on page 32 for information about processing AFP raster fonts as images.
- The PDF file must not contain any passwords or encryption; therefore, none of the functions described in "AFP2PDF Transform security" on page 23 should be used when this parameter is set.

PfmPfb_Directory=*directory*
Specifies the path location of Adobe Type 1 outline font files that the transform uses when embedding fonts inside of the PDF document. When simple font substitution is not acceptable in the PDF output, it is possible to embed a custom Type 1 font inside the PDF document for better results.

Note: Placing Type 1 font files in this directory does not mean they are automatically placed inside of the PDF file. For a font to be embedded, it must be mapped using the transform font definition files. See "Mapping the AFP font to the embedded Type 1 font" on page 35 for more information.

Preserve_CMYK=True | False
Indicates whether all AFP images are converted to the RGB color space in the PDF output. Setting this parameter to **True** preserves AFP images defined with the CMYK color space so they are not converted to RGB and uses a compression in the PDF that does not cause any loss of color quality.

Printer_Resolution=*n*
Specifies the printer resolution to be used in GOCA. The default value is **300**.

ResourceDataPath=*directory***[;***directory***...]**
Specifies the directories that the transform uses to search for AFP resources. You can specify multiple directories, but they must be separated with a semicolon (;). See "Using AFP resources" on page 22 for more information.

Shade_RGB=*red***,***green***,***blue*
Specifies the intensity of the red, green, and blue colors when generating the color of shaded areas. The color values are in the range of 0.0 to 1.0, with "0.0" indicating black and "1.0" indicating white. For example: `Shade_RGB=0.5, 0.5, 0.5`.

See "Substituting AFP shaded images with colored areas" on page 43 for more information about shaded areas.

Show_Outline=True | False
Indicates whether the outline window is displayed. If an AFP document contains index data, the transform converts this index information into PDF outline and bookmark functions. If the output PDF file contains any bookmark information, the outline window is always displayed when viewed with Adobe Acrobat. This parameter can be set to **False** so the outline window is not displayed.

Show_Pageids=True | False
Indicates whether the Page Identifier values are displayed in the bookmark pane.

Static_Paper_Center=True | False
Indicates whether the data in the AFP logical page is centered relative to the static dimensions. This parameter can only be used when Static_Paper_Length and Static_Paper_Width are specified.

Static_Paper_Length=*n*
Overrides the AFP logical page size used by default for the page or media dimensions represented in the PDF. Using units of 72 units per inch, a specific paper length can be specified for the entire document.

Static_Paper_Width=*n*
Overrides the AFP logical page size used by default for the page or media dimensions represented in the PDF. Using units of 72 units per inch, a specific paper width can be specified for the entire document.

Subject=*text*
Specifies 1 to 62 characters for the Subject entry in the Info Dictionary of the output PDF file. This information is displayed to the user if the "document properties" function is selected in Adobe Acrobat. Enclose the text in double quotes if the value contains blanks. For example: `Subject="Advanced Function Presentation"`.

Title=*text*
Specifies 1 to 62 characters for the Title entry in the Info Dictionary of the output PDF file. This information is displayed to the user if the "document properties" function is selected in Adobe Acrobat. Enclose the text in double quotes if the value contains blanks. For example: `Title="InfoPrint Solutions Company Advanced Function Presentation"`.

Transform_All_Subgroups=True | False
Indicates whether the transform converts all of the subgroup formatting specified in the AFP form definition resource. By default, only the first subgroup is processed by the transform.

| **TrueType_Directory=***directory*
| Specifies the path location of TrueType outline font files that the transform
| searches for when the AFP data references TrueType fonts. By default, this
| directory is in the transform's `\font\TrueType` subdirectory.

| If the TrueType font objects are inline with the AFP data, the TrueType fonts
| are written to a file in the specified directory. On UNIX systems, the directory
| must have the appropriate file system permissions to allow the files to be
| written.

| See "Mapping AFP TrueType fonts" on page 37 for more information.

UDC_File
Reserved for future use.

| **UDC_Range=***low1*,*high1*[,*low2*,*high2*][,*low3*,*high3*][,*low4*,*high4*]
| Specifies one to four section ranges that the transform can search when
| processing user defined characters for double-byte character set (DBCS) text.
| The ranges restrict which DBCS sections are searched, which causes the
| transform to run more efficiently.

| For example, if the user defined characters are present in sections 129, 130, and
| 131, the parameter can specify these low and high sections: `UDC_Range=129,131`.

Use_AFP_Metrics=*n*
: Specifies a number that tells the transform to use the AFP font metrics instead of the mapped font metrics. For example, `Use_AFP_Metrics=255`. See “Using custom AFP raster font files” on page 32 to determine how the AFP font metrics can be used instead of the mapped font metrics.

Use_ICU=True | False
: Indicates whether the transform uses International Components for Unicode (ICU) functions instead of standard **iconv** functions to convert text data that requires more than one byte to represent a single character, such as these Asian languages: Chinese, Japanese, and Korean.

Use_Unicode=True | False
: Indicates whether the text in the PDF uses Universal Character Set, 2–byte (UCS-2) encoding. This parameter only applies to Asian languages, such as Chinese, Japanese, and Korean.

Vertical_Offset=*n*
: Applies a vertical offset to all the data on the page relative to the paper or media represented in the PDF. The units of this setting are in 1440 units per inch.

split_afp2pdf command

The **split_afp2pdf** command takes an indexed AFP print file, splits it up into separate statements, and invokes the AFP to PDF transform. The input AFP data must be divided into separate page groups (statements) and contain AFP index values for each group. This type of data is created by the AFP Conversion and Indexing Facility (ACIF) or an equivalent function.

The **split_afp2pdf** command creates a separate PDF output file for each statement. The name of the output file corresponds to at least a single associated index value. For example, if the index field, ACCOUNT, is selected, each output file is named with the actual account ID (such as `123456.pdf`).

This command invokes the AFP2PDF Transform for each statement so the same configuration and setup for the transform program applies. The `split_afp2pdf` module must be placed in the same directory where the AFP2PDF Transform files are installed.

Syntax

The syntax for the **split_afp2pdf** command is:

```
►►──split_afp2pdf──┬──────────┬──┬─────────┬──┬──────────┬──┬──────────────┬──────────►
                   └─-d──dir──┘  └─-e──pw──┘  └─-f──def──┘  └─-g──code_pg──┘

►──┬───────────────┬──┬────────────┬──┬─────────────┬──┬─────────────┬────────────────►
   └─-h──fontpath──┘  └─-i──index──┘  └─-i2──index──┘  └─-i3──index──┘

►──┬─────────────┬──┬──────────────┬──┬──────────────┬──┬───────────┬──┬────┬─────────►
   └─-ix──index──┘  └─-ix2──index──┘  └─-ix3──index──┘  └─-k──path──┘  └─-l─┘

►──┬──────────────┬──┬────┬──┬──────────────┬──┬────────────┬──┬──────────────┬──────►
   └─-m──mapfile──┘  └─-n─┘  └─-o──optfile──┘  └─-p──codes──┘  └─-r──resfile──┘
```

```
►──┬──────────┬──┬────────┬──┬────────────┬──afpfile──────────────────────►◄
   └─-t──turn─┘  └─-u──pw─┘  └─-v──value──┘
```

Parameters

-d *dir*
: Specifies the directory for all of the output files. If this parameter is not specified, the output files are placed in the current directory.

-e *pw*
: Specifies an alphanumeric password that gives permission to change document security settings (also known as a *master* or *owner* password). For example, if the printing function in the PDF display has been restricted (`-p p`), you must supply a password to override the setting. The same password is applied for each output file.

-f *def*
: Specifies the fully qualified file name of the form definition resource that is used on each statement when transforming the AFP document.

-g *code_pg*
: Specifies the code page identifier (ID) to be used when interpreting the index information in the AFP file. By default, code page 500 is used to interpret all index fields. See "Code page map file" on page 30 for more information .

-h *fontpath*
: Specifies the location of the font definition files when a code page ID is specified with the -g parameter. The -h parameter should be used if the font definition files are not located in the `\font\maps` subdirectory.

-i *index*
: Specifies the index field used for generating the output file names. For example, if `-i RouteID` is specified, the output file name `ABC` is generated if a statement has a routing ID of `ABC`.

  The user must know which index fields are available for the AFP data and must pick a field that is unique for each statement. For example, a single index field, such as ZIPCODE, might not be have unique values for all statements. In that case, statements with the same zip code would have the same file name causing an overwrite of the output file.

  If this parameter is not given, the first index field found in the AFP data is used. If an index field name not available in the AFP data is given, an error message is issued and the program ends.

-i2 *index*
: Specifies the second index field used for generating the output file names. This parameter should only be used if the -i parameter is specified. The value of this index field is concatenated to the end of the first index field and a hyphen (-) is used as a separator. For example, if `-i RouteID -i2 AccountNo` is specified, the output file name `ABC-1234567` is generated if a statement has a routing ID of `ABC` and an account number of `1234567`.

-i3 *index*
: Specifies the third index field used for generating the output file names. This parameter should only be used if the -i and -i2 parameters are specified. The value of this index field is concatenated to the end of the second index field and a hyphen (-) is used as a separator. For example, if `-i RouteID -i2`

`OfficeNo -i3 AccountNo` is specified, the output file name of `ABC-XYZ-1234567` is generated if a statement has a routing ID of `ABC`, an office number of `XYZ`, and an account number of `1234567`.

-ix *index*
: Specifies the index field used for generating the output file names. This parameter is the same as the -i parameter but is used when the index name is specified in hexadecimal notation.

-ix2 *index*
: Specifies the second index field used for generating the output file names. This parameter is the same as the -i2 parameter but is used when the index name is specified in hexadecimal notation.

-ix3 *index*
: Specifies the third index field used for generating the output file names. This parameter is the same as the -i3 parameter but is used when the index name is specified in hexadecimal notation.

| **-k** *path*
| Specifies the location of the `split_afp2pdf` module files. The -k parameter
| should be used if the `split_afp2pdf` module is invoked from a different
| directory than the directory where the AFP2PDF Transform files are installed.

-l Generates a linearized PDF file which reorganizes the data for more efficient processing in a network environment. Also known as Fast Web View PDF documents, files generated with this option make it possible to view very large documents without long download delays. To use this parameter, verify that the Web server or Web application sending this PDF data over the network provides the page-by-page downloading or "byte-serving" function.

-m *mapfile*
: Specifies the location and name of the image map configuration file that is used by the transform.

-n Generates a different PDF output file name if the file already exists. For example, the PDF output file `ABCD__2.pdf` is generated if the output file name `ABCD.pdf` already exists.

-o *optfile*
: Specifies the location and name of the options file used by the transform.

-p *codes*
: When used with an owner password (`-e` *pw*), specifies which PDF display functions are restricted. The codes for the display functions, which can be used in any order, are:

a	Add or modify text annotations and interactive form fields.
c	Modify the document contents.
d	Copy text and graphics from the document.
p	Print the document.

-r *resfile*
: Specifies the fully qualified file name of the AFP resource group file to be used when transforming the AFP file.

-t *turn*
: Specifies the rotation value to use when transforming the file. Valid values are

0 , **90**, **180**, and **270**. Some AFP files might have already been formatted with a rotated orientation; therefore, you must use this parameter to align the text in an upright position.

-u *pw*
: Specifies an alphanumeric password that gives permission to open the PDF document (also known as a *user* password).

-v *value*
: Specifies the index field value where the statement conversion process begins. This parameter lets the conversion start someplace other than the first statement at the beginning of the file. The -i index setting must also be specified if this parameter is used. For example, if `split_afp2pdf -i ACCOUNT -v 123456` is specified, all statements in the AFP data are skipped until the account 123456 is encountered, then the transform process resumes.

afpfile
: Specifies the AFP input file that is to be transformed to PDF. This parameter is required.

Return codes

When the **split_afp2pdf** command runs, you see one of these return code values:

0 Successful completion of the statement split and transform.

Nonzero
: An error has occurred.

split_afp2pdf generates a log file containing the list of output files generated. The log file is placed in the same directory as the output files. If an error occurred during the conversion process, this log could be used to determine the last statement processed and used as the restarting value. This log also contains any warning messages created during the conversion. For example, if an output file was overwritten, it would be indicated as a warning message in this log file.

Using AFP resources

The AFP resources used by AFP2PDF Transform include:

- Page segments
- Overlays
- Form definitions
- Single-byte character set raster font files

Currently, AFP2PDF Transform does not process double-byte character set (DBCS) or outline AFP font files. If the program encounters resources of this type, they are ignored. To display text, the transform first tries to use the fonts available within Adobe Acrobat. Font definition files that map the standard AFP fonts to Acrobat fonts for PDF are provided. If your AFP document uses AFP fonts that you have customized or created, you must map these fonts. For information about mapping AFP fonts, see Chapter 4, “Mapping fonts,” on page 25.

The page segment, overlay, form definition, and font file resources can be passed to the transform from these locations:

Inline resource group
: The AFP resources needed by the AFP data file are combined into a logical resource library for the document. This resource group is contained in the AFP file along with the AFP document.

External resource group

The AFP resources needed by the AFP data file are combined into a logical resource library for the document and are passed to the transform in as a separate file.

When using the **afp2pdf** command, this resource file is specified with the -r parameter. See "Syntax" on page 7 for more information.

Resource directories

AFP resource files can be placed in specific directories that the transform program searches for when converting a document. The user can specify multiple directories and the directories are searched in the order that they are given.

By default, resources are placed in a `\resource` subdirectory where the transform code modules were installed. You can specify other directories with the ResourceDataPath parameter in the transform options file. See "AFP2PDF Transform options file" on page 10 for more information.

You can also associate a file extension for the page segment and overlay resources. Using the PageSegExt and OverlayExt parameters in the transform options file, the given file extension is used when searching for the resource. For example, if `PageSegExt=*.PSG` is set in the options file, and the page segment resource called from the AFP data file is `S1PAGSEG`, the transform searches for the file named `S1PAGSEG.PSG` in the resource directories.

Note: The name of the resource file is case-sensitive on the UNIX servers and it must match the name of the resource that is specified in the AFP data file.

If an AFP resource is located in multiple places, the transform program uses this search order:

1. Internal resource group
2. External resource group
3. Resource directories specified with the ResourceDataPath parameter in the options file
4. The `\resource` subdirectory where the transform modules were installed

AFP2PDF Transform security

Protecting the contents of the PDF document is accomplished with encryption. This PDF security feature is supported by the AFP2PDF Transform and follows the password features supported within the Adobe Acrobat product. A PDF document can have two levels of password protection: a "document open" password and a "permissions" password.

When a document open password (also known as a *user* password) is used, any user who tries to open the PDF document is required to supply the correct password.

Encrypted PDF in the transform is also tied to restricting certain functions when displayed in Adobe Acrobat. If any display functions are restricted, a permissions password (also known as an *owner* or *master* password) is required. Any user needing to override a restricted function must supply the correct permissions password. See "Parameters" on page 7 for information about using the -a and -e parameters to restrict display functions.

Both types of passwords can be set for a document. If the PDF document has both types of passwords, it can be opened with either password.

Only Adobe software fully supports and respects these settings; users of third-party PDF-enabled programs might be able to bypass some of the restrictions.

Chapter 4. Mapping fonts

The AFP2PDF Transform must map the AFP fonts your document was created with to fonts that can be displayed with Adobe Acrobat. The AFP2PDF Transform uses font definition files that are loaded into the `font` subdirectory during installation.

| This chapter describes font definition files, the process for mapping fonts, custom AFP raster font files, embedded Type 1 fonts, custom font metric files, and AFP TrueType fonts.

| **Note:** The directory and file examples in this chapter are specified for the Windows environment. To use these examples in a UNIX environment, use the UNIX file naming convention for any file name. For example, `\font\maps` in Windows is `/font/maps` in a UNIX environment, or the input AFP file `c:\documents\afpdoc.afp` in Windows would be `/documents/afpdoc.afp` in a UNIX environment.

Files supplied for mapping fonts

Table 1 lists the AFP2PDF Transform font support files and the subdirectories in which they are installed. (The directory is that in which the AFP2PDF Transform was installed.)

Table 1. Font files and subdirectories

File	File name	Subdirectory	Description
Alias file	alias.fnt	\font	Maps the font type families to PDF font name and optionally specifies the font metric file to use during the transform.
Character set definition file	csdef.fnt	\font	Defines AFP character set attributes, such as point size. It also maps the font character set to the font global identifier.
Coded font file	icoded.fnt, coded.fnt	\font	Specifies which AFP code page and AFP font character set make up the coded font.
Code page definition file	cpdef.fnt	\font	Maps each AFP code page to indicate which code page map file to use for the AFP2PDF Transform.
Code page map file	*cpgid*.cp	\font\maps	Defines character identifier mappings. It matches the AFP code page character identifiers and their hexadecimal code points with a corresponding character identifier and ASCII code point available in the PDF environment. The default is Code Page 1252.
Font mapping file	afpfont.fnt	\font	Defines the fonts where the raster character images from the AFP font files are extracted and placed as images in the output PDF.

Table 1. Font files and subdirectories (continued)

File	File name	Subdirectory	Description
Font metric information file	*font*.AFM	\font\AFM	Contains the font metric information, which is the dimension of each of the characters (optional).
TrueType outline font files	*font*.ttf	\font\TrueType	Contains the default location for the TrueType font files (*.ttf) when TrueType fonts are referenced in AFP files.
Type 1 outline font files and custom font metric files	*font*.pfb *font*.pfm Custom-Metrics-*xx*.met	\font\Type1	Contains the default location for the Type 1 font files (*.pfb, *.pfm) to be embedded. Can also hold metric information for custom fonts (optional).

Coded Font File

The coded font file (`coded.fnt` or `icoded.fnt`) maps AFP coded fonts to their AFP character sets and code pages. Table 2 describes the two coded font files that can be used with AFP2PDF Transform.

Table 2. Coded font files

Coded font file name	Description
coded.fnt	Contains user-defined coded fonts. This file is optional, but must be placed in the `\font` subdirectory.
icoded.fnt	Contains standard definitions for approximately 2500 coded fonts supplied by InfoPrint Solutions Company.

If a `coded.fnt` file exists in the `\font` subdirectory, AFP2PDF Transform searches it first for the coded fonts used in an AFP file. Figure 1 shows an example of the contents of the `coded.fnt` file.

```
X?A155N2=C?A155N1,T1DCDCFS
X?AE10=C?S0AE10,T1S0AE10
X?GT10=C?D0GT10,T1D0BASE
X?ST15=C?D0ST15,T1D0BASE
X?A0770C=C?A07700,T1GI0361
X0T0550C=C0T05500,T1DCDCFS
```

Figure 1. Example of a coded.fnt file

Syntax rules:

- A question mark (?) can only be used as the wildcard character for the second character in the coded font name and the character set name. This allows all the character rotations of the coded fonts to be handled with one entry while searching.

  Note: A sequential search is performed for the coded font, and the first match is used (including the wildcard character).
- After the coded font name, the character set name must be listed first, followed by the code page name.
- The character set and code page must be separated by a comma.

Character set definition file

The character set definition file (`csdef.fnt`) specifies the character set attributes and font global identifier of the font. It is split into two sections, one for character sets (CHARSET) and one for font global identifiers (FGID). The CHARSET section lists each AFP font character set and its corresponding attributes. Figure 2 shows an example of the CHARSET section in the `csdef.fnt` file:

```
[CHARSET]
 ;charset = fgid, height, width, strikeover, underline
 C?H200A0=2304,110,73,0,0
 C?H200D0=2304,140,93,0,0
 C?N200B0=2308,120,80,0,0
 C?4200B0=416,120,144,0,0
 C?D0GT15=230,80,96,0,0
 C?A155A0=33207,110,73,0,0
 C?A175A0=33227,110,73,0,0
 C?T055D0=4407,140,93,0,0
 C?T17500=4555,100,67,0,
 C?T17560=4555,60,40,0,0
 DEFAULT=2308,80,0
```

Figure 2. CHARSET section of csdef.fnt file

Table 3 describes the attributes and values for CHARSET.

Table 3. Attribute values for CHARSET

Attribute	Values	Shipped default	Description
fgid	An FGID in one of these ranges: • 3840 to 4096 • 65260 to 65534	2308	A unique font global identifier (FGID) value, which identifies the type family, typeface, and sometimes the point size of the character set. This can be a predefined FGID or your own FGID.
height	1 to 990	80	The vertical size of the character set (minimal baseline-to-baseline value) expressed in tenths of a point. For example, a 9-point font would have a height of 90.
width	0 to 99 (currently ignored)	0	The average horizontal size of the characters in 1440th of an inch. Currently, 0 is always used because an appropriate font width is determined based on the height of the font.
strikeover	1 = YES 0 = NO	0	A font whose characters all have a line, parallel to the character baseline, placed over the middle of the character.
underline	1 = YES 0 = NO	0	A font whose characters all have a line underneath the character.

The FGID section lists each font global identifier and its corresponding attributes. Figure 3 on page 28 shows an example of the FGID section in the `csdef.fnt` file:

```
[FGID]
 ;fgid = familyname, style, weight, italic
 230=Gothic, MODERN,MED,0
 416=Courier,MODERN,MED,0
 2304=Helvetica,SWISS,MED,0
 2308=TimesNewRoman,ROMAN,MED,0
 4407=SonoranSerif,ROMAN,MED,0
 4555=SonoranSerif,ROMAN,BOLD,1
 33207=SonoranSansSerif,SWISS,MED,1
 33227=SonoranSansSerif,SWISS,BOLD,1
```

Figure 3. FGID section of csdef.fnt file

Table 4 describes the attributes and values for FGID.

Table 4. Attribute Values for FGID

Attribute	Values	Shipped default	Description
familyname	Font family name reference	Times New Roman	An outline font name or an AFP type family name; "familyname" is the same as "type family" in AFP fonts and "typeface name" available in the PDF environment.
style	SWISS, ROMAN, SCRIPT, MODERN, DISPLAY	ROMAN	A type of character face or specific characteristics of the font. **Notes:** 1. SWISS is a proportionally spaced, sans serif font. 2. ROMAN is a proportionally spaced, serif font. 3. SCRIPT is a fixed-pitch font designed to look like handwriting. 4. MODERN is a fixed-pitch, sans serif or serif font. 5. DISPLAY is a decorative font.
weight	LIGHT, MED, BOLD	MED	The degree of boldness of a typeface caused by different thickness of the strokes that form a graphic character.
italic	1=YES 0=NO	0	A font with right-slanting characters.

Sytax rules:

- A comma must separate attributes.
- A question mark (?) can only be used as the wildcard character for the second character in the character set name. This allows all the character rotations of the coded fonts to be handled with one entry while searching.

  Note: A sequential search is performed for the character set, and the first match is used (including the wildcard character).
- The CHARSET section must come before the FGID section in the file.
- In the CHARSET section of the file, only the “fgid” and “height” attributes are required.
- In the FGID section of the file, only the “familyname” and “style” attributes are required.

- If you define a default character set in the file, it must be the last entry in the CHARSET section.
- If you add your own AFP font character set to the CHARSET section, you must assign it a font global identifier. If the new character set has the same “familyname”, “style”, “weight”, and “italic” attributes as an existing character set, you can use the same font global identifier; otherwise, you must add a unique font global identifier to the FGID section.

Code page definition file

The code page definition file (`cpdef.fnt`) maps the AFP code page name to its code page global identifier (CPGID) and, optionally, to an encoding classification and an encoding type for Asian languages. The section header, CODEPG, is followed by a list of AFP code pages and their attributes. The first attribute in each line is the AFP code page global identifier that maps to a code page map file (see “Code page map file” on page 30 for more information about mapping code pages). The second attribute is the encoding classification that you decide is the best match for your AFP code page. The third attribute is the single-byte character set (SBCS) or double-byte character set (DBCS) encoding type that is needed for some Asian languages. The last line gives the default attribute values to be used when a default is required. Figure 4 shows an example of the contents of the `cpdef.fnt` file.

```
[CODEPG]
;codepage = cpgid,wincp,entype
T1DCDCFS=1003,ANSI
T1DEBASE=2058,ANSI
T1D0BASE=2063,ANSI
T1D0GP12=2085,ANSI
T1GI0395=2079,ANSI
T1GPI363=2066,SYMBOL
T1V10037=37,ANSI
T1V10273=273,ANSI
T1000290=290,ANSI
T1000310=310,ANSI
T1000423=423,ANSI
T1000905=905,ANSI
P1D0BASE=456,OEM,SBCS
DEFAULT=361,ANSI
```

Figure 4. CODEPG section of cpdef.fnt file

Table 5 describes the attributes and values for the AFP code pages in the code page definition file.

Table 5. CODEPG attributes

Attribute	Value	Shipped default	Description
cpgid	A CPGID in the range of 65280 to 65534	361	An AFP-defined code page global identifier (CPGID), your own defined CPGID, or any other CPGID not already being used within the file

Table 5. CODEPG attributes (continued)

Attribute	Value	Shipped default	Description
wincp	ANSI, SYMBOL, OEM, or NONSTD KYUJIS (Japanese) BIG5 (Traditional Chinese) GB (Simplified Chinese) KSC (Korean) IDENTITY (Unicode)	ANSI	PDF encoding
entype	SBCS or DBCS		SBCS or DBCS encoding type for Asian language definitions

Syntax rules:

- A comma must separate attributes.
- Only the first attribute, “cpgid”, is required.
- If you create your own code page, you must assign it a unique code page identifier. Leading zeros are not valid. (You can use a predefined code page global identifier, but only if the character-to-hexadecimal code mapping is the same for your code page.)
- If you define a default code page in the file, it must be the last entry in the file.

Code page map file

AFP2PDF Transform provides one code page map file for each AFP code page supplied with Print Services Facility™ (PSF) and the Data1 and Sonoran licensed programs. These files are installed in the `\font\maps` subdirectory. The file is named for its code page global identifier (CPGID) and has a file extension of “.cp” (for example, if “2063.cp” is the file name for the T1D0BASE code page map; its CPGID is “2063”). Each file contains the character identifiers (and associated EBCDIC hexadecimal code points) for an AFP code page and maps them to character identifiers (and associated ASCII code points) for an ANSI or SYMBOL PDF encoding.

Figure 5 on page 31 shows an example of the contents of the code page map file “395.cp” for the “T1000395” code page mapped to the ANSI encoding.

```
;T1000395 to ANSI
 SP010000 40  SP010000 20
 LA150000 42  LA150000 E2
 LA170000 43  LA170000 E4
 LA130000 44  LA130000 E0
 SP180000 8B  SP180000 BB
 SM560000 8C  SM560000 89
 SA000000 8D  SP100000 2D
 LI510000 8E  NOMATCH 00
 LF570000 8F  NOMATCH 00
 SM190000 90  SM190000 B0
 LJ010000 91  LJ010000 6A
 LF510000 A0  NOMATCH 00
 ;;;;;;;; ;   SD150000 5E
 ;;;;;;;; ;   SD130000 60
 ;;;;;;;; ;   LT630000 FE
 /*
```

Figure 5. Code page map file example

Syntax rules:

- Blanks must separate character identifiers.
- NOMATCH means there is not a matching character in the output character set.
- The NOMATCH hexadecimal code of "00" is mapped to the undefined code point. When a document contains a character that does not exist in the output character set, that character cannot be displayed. If the character has not been remapped in the code page map file or the alias file, the undefined code point character is displayed as a substitute.
- The string of semicolons (;;;;;;;;;) means this line is ignored as a comment. It also indicates that the output code page contains a character that does not exist in the AFP code page. The code point for a character not found in the AFP code page can be used for replacing NOMATCH characters.
- If the input code point maps to "NOMATCH 00", and the corresponding AFP code page and character set resources are available (inline or in a resource directory), the transform extracts the raster character from the AFP font and places it as an image in the PDF output.

Alias file

The alias file (`alias.fnt`) lists the font metric file name and the font family name aliases in the FONT section. Font family name aliases let you change all of the requested instances of a font family name (as defined in the character set definition file) to another font family name.

Figure 6 shows an example of how the `alias.fnt` file is used with the AFP2PDF Transform to change all requests for the SonoranSerif font to requests for the Times font (which is one of the base fonts available in the Adobe Acrobat Viewer).

```
[FONT]
 ; ***** Requested font = font name,Font metric/AFM filename (or 'NULL' for not used) *****
 SonoranSerif=Times, NULL
```

Figure 6. Alias file example

Syntax rules:

- If multiple mappings are listed in the file for the same family name, only the first match is used.

- The alias file is processed sequentially. Items within the alias file are not chained. That is, if “Century Schoolbook” is set equal to “Times,” and “Times” is set equal to “Times New Roman”, then “Century Schoolbook” is not set equal to “Times New Roman”.
- Blanks in family names are treated as characters. For example, “New Century Schlbk” is not the same font as “NewCenturySchlbk”.

Process for mapping fonts

If your document uses an AFP font that is not listed in the font definition file, if you have modified the AFP fonts, or if you have created your own AFP fonts, you must edit the font definition files to add the fonts so documents using those fonts display correctly with AFP2PDF Transform. For example:

- If you created a new coded font (or renamed one), you need to define the coded font in the coded font file (`icoded.fnt` or `coded.fnt`).
- If you created a new character set, you must define it in the character set definition file (`csdef.fnt`).
- If you created a new code page, you must define it in the code page definition file (`cpdef.fnt`).
- If you have created a new code page or modified a code page by moving characters, you need to create a new code page map file (*`cpgid`*`.cp`).

If you have only modified an existing font component, such as deleting code points in the code page, you might not need to edit some of the definition files.

After determining which font files you need to modify, follow these steps to map your fonts:

1. Gather the information needed to define the fonts in the font definition files.
2. Make backup copies of any of these font definition files you plan to modify so you have the original file in case the modified copy becomes corrupted:
   ```
   afpfont.fnt
   alias.fnt
   coded.fnt
   cpdef.fnt
   csdef.fnt
   ```
3. Substitute any nonmatching characters in the code page map file. See “Code page map file” on page 30 for information about code page map files.
4. Edit the `cpdef.fnt` file and add your code page name, code page identifier, and the best matching encoding classification for the fonts you are using.
5. If you have created a new character set, edit the `csdef.fnt` file and add your character set name in the CHARSET section. Specify the correct attributes for your font in the `csdef.fnt` file. Add the appropriate information in the FIGID section of the file if you are naming a new font global identifier.
6. If you have created a coded font, create or edit the `coded.fnt` file and add your coded font.
7. If any of the AFP raster fonts need to be used, list the corresponding AFP coded fonts, code pages, and character sets in the `afpfont.fnt` file.

Using custom AFP raster font files

Custom AFP raster fonts can be mapped using AFP2PDF Transform. To do this, the AFP resources, coded fonts, code pages, and character sets, must be either inline or available in the resource directories. This is the order that the transform checks for resources to map the raster fonts:

1. The transform looks for an entry in the `afpfont.fnt` file that maps the resource set. The entry can include limited wildcards: a question mark (?) can be used to represent any character in one position; an asterisk (*) can be used to indicate one or more characters. For example, the entry C?T175* can represent any of these values: C0T17500, C0T17560, and C1T17500. This is the hierarchy that is followed:
   a. If the resource is allowed and it is inline, it is used.
   b. If the resource is not found, the second character in the name is replaced with "0" and the inline resource is checked again.
   c. If the resource is not inline, a file is looked for in the resource paths (the ResourceDataPath parameter in the transform options file can be used to indicate the location of these paths; the default is the `\resource` subdirectory in the current working directory).
   d. If the resource is still not found, the second character in the name is replaced with "0" and the resource paths are searched again.
   e. If the resource files have an extension, it can be specified using the FontExt parameter in the transform options file (for example, “FontExt=*.240” searches for fonts with the extension “.240”).
2. If there is no entry in the `afpfont.fnt` file or if the resource is not found, the `icoded.fnt`, `cpdef.fnt`, and `csdef.fnt` files are checked to see if the resource is to be mapped. If the resource is found, it is mapped. If **Enable_Auto_Font_Image=True** in the transform options file, the resources are checked as follows:
   a. If the resource is inline, it is used.
   b. If the resource is not found, the second character in the name is replaced with "0" and the inline resource is checked again.
   c. If the resource is not inline, a file is looked for in the resource paths (the ResourceDataPath parameter in the transform options file can be used to indicate the location of these paths; the default is the `\resource` subdirectory in the current working directory).
   d. If the resource is still not found, the second character in the name is replaced with "0" and the resource paths are searched again.
   e. If the resource files have an extension, it can be specified using the FontExt parameter in the transform options file (for example, “FontExt=*.240” searches for fonts with the extension, “.240”).
3. If the resources still have not been found, the defaults are used.

Note: If a file is using:
- A mapped code page and a custom character set, the font is converted to image data.
- A mapped character set and a custom code page, the font is not converted to image data. The custom code page is mapped using a map file.

Figure 7 on page 34 defines raster fonts in a font mapping file. In this example, code pages T1K99MAP, T1K99MCP, and T1KNIC4 are mapped. Character sets C?CONWAY, C?K99MAP, C?K99MCP, and C?WSMITH are also mapped.

```
[CODEDFNT]
 ;Coded Font
 ;X?*
 [CODEPG]
 ;Code Page
 ;T?*
 T1K99MAP
 T1K99MCP
 T1KNIC4
 [CHARSET]
 ;Character Set
 ;C?*
 C?CONWAY
 C?K99MAP
 C?K99MCP
 C?WSMITH
```

Figure 7. Font mapping file example

The font mapping file can also be used in these situations:

- Some special characters in a font, such as a pointing finger, are not represented in the Type 1 font that is being used. In this case, AFP raster fonts can be used for some characters while the fonts defined in the font definition files, such as `csdef.fnt` and `cpdef.fnt`, can be used for others characters. For example, `C?H00040 84,98,C1` uses AFP raster fonts for code points X'84', X'98', and X'C1' and the fonts defined in the font definition files for all other characters.
- When the Use_AFP_Metrics parameter is specified in the transform options file, such as `USE_AFP_METRICS=255`, and a character set, such as `C?H00040 FF`, is specified in the font mapping file, the AFP font metrics are used for character placement when the font defined by `C?H00040` in the `csdef.fnt` file is used.
- The first two examples can be combined to use the AFP font metrics and some AFP raster fonts.

Figure 8 shows entries in the transform options file that define resource paths. In this example, "c:\res1" is searched first for resources with an extension of ".240" and then "c:\res2" is searched.

```
ResourceDataPath='c:\res1\;c:\res2'
 FontExt=*.240
```

Figure 8. Transform options file example

Embedding Type 1 fonts

When you use a custom AFP font within an AFP document, it might be necessary to generate an equivalent Type 1 outline font. You can configure the transform program to embed this Type 1 outline within the output PDF file for proper display.

The Type 1 font must be in binary (PFB, PFM) format. If a font is in ASCII (AFM, PFA) format, you must convert it before it can be used with the transform. Type 1 fonts typically consist of a printer font binary (PFB) and a printer font metrics (PFM) file. The PFB file contains the mathematical descriptions (in the PostScript® language) for each character. The PFM file contains the font metrics needed for the display of the characters.

For example, assume a Type 1 font is made up of the files, `CustFont.pfb` and `CustFont.pfm`. The file extensions of ".pfb" and ".pfm" must be lowercase for the transform to correctly identify these files. The font file name, CustFont, must be the same for both the .pfb and .pfm files. For the transform to use these files, these must be defined:

- File location
- Rules for when the files should be substituted for a specific AFP font

File location

You must place all Type 1 font files, including all font files that can be used for embedding fonts in the PDF file, in a single directory. By default, the transform looks in the `\font\Type1` subdirectory in the same directory where the AFP2PDF Transform files are installed. The PfmPfb_Directory parameter in the transform options file can be used to define a directory that is different from the default. See "AFP2PDF Transform options file" on page 10 for more information.

Mapping the AFP font to the embedded Type 1 font

The embedded Type 1 font name is specified in the `alias.fnt` font mapping configuration file. Under the FONT section, a font name and font metric file name are specified for the font:

```
[FONT]
 font=font_name,font_metric_name
```

Where:

- *font* defines the name by which this font is known and is referred to by the other font configuration files. For example, `Custom1`.
- *font_name* specifies the name of the font, such as `CustFont`. This name is case-sensitive and must match the file name of the PFB and PFM files.
- *font_metric_name* indicates the file name for font metric information. If it is set to NULL, it indicates that extra font metric information is not used.

In this example, if the AFP font is mapped to "Custom1", the Type 1 font, "CustFont", is embedded inside the output PDF file and the appropriate text uses this font for the display:

```
[FONT]
 Custom1=CustFont,NULL
```

See Chapter 4, "Mapping fonts," on page 25 for more information about mapping fonts.

Using custom font metric files

AFP2PDF Transform can be configured with custom font metric files to control the placement of individual characters and to aid in text alignment. AFP2PDF Transform uses the default fonts available in the PDF display application (for example, Adobe Acrobat) and applies special character widths as specified in the font metric file.

Font metric files should be set up to use the default PDF WinANSI encoding (code page 1252); they cannot be used for double-byte text (Asian languages). The files are located in the `\font\Type1` subdirectory and are named `Custom-Metrics-`*x*`.met`, where *x* is 1 to 25.

To configure the transform, the name of the font metric file is specified as the first parameter in the `alias.fnt` file. The second parameter, specifying the Adobe Font Metric (AFM) file name, is set to NULL. Figure 9 shows an example of the `alias.fnt` file with entries that define custom font metric files to the transform.

```
;**** Requested font=font name,Font metric/AFM file name (or 'NULL' for not used) *****
Font1=Custom-Metrics-1,NULL
Font2=Custom-Metrics-2,NULL
;******* End User-defined/Custom names *******
```

Figure 9. Custom font metric files defined in the alias.fnt file

The format of custom font metric files follows the AFM specification, including two additional parameters that describe the font used for the display:

Flags Specifies various characteristics of the font. Values are bit positions from 1 to 32. Table 6 shows the defined bit position values.

Table 6. Defined values for the Flags parameter

Value	Font type	Description
1	FixedPitch	All characters have the same width (as opposed to proportional or variable-pitch fonts, which have different widths).
2	Serif	Characters have serifs, which are short lines drawn at an angle on the upper and lower end of character strokes (as opposed to sans serif fonts, which have no serifs).
6	Nonsymbolic	The font uses the Adobe standard Latin character set or a subset of it.
7	Italic	Characters are slanted to the right.
17	AllCap	The font contains no lowercase letters; typically used for display purposes such as titles or headlines.
18	SmallCap	The font contains capital letters in the same style as the normal capital letters in a font, but approximately the size of the lowercase letters.
Note: All other bit positions are reserved and must be set to 0.		

StemV
The width, measured in the x direction, of the dominant vertical stems of characters in the font.

Figure 10 on page 37 shows an example of an custom font metric file called `Custom-Metrics-1.met`.

```
FontName C0CUSTOM
; Desciption - 'Custom Font 1'
Ascender 924
CapHeight 720
Descender -216
Flags 32
FontBBox -47 -204 996 924
ItalicAngle 0
StemV 90
XHeight 720
StartCharMetrics
C 32 ; WX 240 ; N space ;
C 33 ; WX 252 ; N exclam ;
C 34 ; WX 408 ; N quotedbl ;
C 35 ; WX 480 ; N numbersign ;
C 36 ; WX 480 ; N dollar ;
C 37 ; WX 756 ; N percent ;
C 38 ; WX 552 ; N ampersand ;
C 39 ; WX 240 ; N quotesingle ;
...
EndCharMetrics
```

Figure 10. Example of a custom font metric file

Mapping AFP TrueType fonts

When TrueType fonts are specified on the Map Data Resource (MDR) structured field in AFP documents, they can be mapped using AFP2PDF Transform. The TrueType font file must have a lowercase extension of ".ttf" for the transform to correctly identify the TrueType file. For example, if the MDR specifies "Font1" for the full font name, the transform tries to locate the file named `Font1.ttf`.

TrueType font collections, linked TrueType fonts, or the use of a Resource Access Table (RAT) are not supported.

For the transform to use these files, these must be defined:
- File location
- Converter files for character encoding

File location

You must place all TypeType font files in a single directory. By default, the transform looks in the `\font\TrueType` subdirectory in the same directory where the AFP2PDF Transform files are installed. You can use the TrueType_Directory parameter in the transform options file to define a directory that is different from the default. See "AFP2PDF Transform options file" on page 10 for more information.

Character encoding

The AFP2PDF Transform uses the Unicode character encoding standard to reference characters in TrueType fonts. If the AFP text is encoded in a format other than a standard Unicode encoding (UTF-8 or UTF-16), the text must be converted to Unicode during the transformation process. In that case, new conversion tables that use the International Components for Unicode (ICU) function might be required.

The `\locale` subdirectory is the default location for the ICU binary converter files (*.cnv). You can use the Locale_Path parameter in the transform options file to define a directory that is different from the default. See "AFP2PDF Transform options file" on page 10 for more information.

The binary converter files are based on ICU 3.4 and are different for the system being used. The file name convention is `icudt34`*p*`_ibm-`*cpid*`.cnv`, where *p* denotes the system ID and *cpid* is the code page ID. The values for the system ID are:

l	Little endian, ASCII
b	Big Endian, ASCII
e	Big Endian, EBCDIC

The code page ID is the numerical CPGID that is defined in the `cpdef.fnt` code page definition file (see "Code page definition file" on page 29).

Chapter 5. Mapping AFP images

When an AFP document is transformed, images are identified using parameters that specify the page segment name (if available), the position, and the size of each image. If an AFP page contains images, AFP2PDF Transform creates image information entries in an output file. The output file entries can be copied into an image map configuration file to define and map a particular image.

Mapping images lets you handle AFP images in different ways during the transform process, such as:

- Identifying individual images in your transformed files.
- Removing all or some of the images from your transformed output.
- Substituting all or some of the images with previously generated images in the PDF output with JPEG images.
- Substituting all or some of the images with a solid-colored rectangle. This is especially useful for improving the look of the shaded areas that are defined as images in the AFP data stream.
- Adding an image, which is not part of the AFP data, to the PDF display that models the use of a preprinted form used during printing.
- Storing frequently used images to reduce the size of a PDF file.

The configuration file handles all the transform processing for the images. For example, when the transform program is run against an AFP document and an image is encountered, the program looks for a matching image entry in the configuration file. If an entry is defined that matches the name, position, size, or a combination of the three, the information in the configuration file is used to transform the image. If a matching image entry in the configuration file contains an "empty entry", the image is not generated in the PDF output file.

| **Note:** The file examples in this chapter are specified for the Windows
| environment. To use these examples in a UNIX environment, use the UNIX
| file naming convention for any file name. For example, the input AFP file
| `c:\documents\afpdoc.afp` in Windows would be `/documents/afpdoc.afp` in a
| UNIX environment.

Creating the image map configuration file

To map images for your AFP files, you must create an image map configuration file. The best way to do this is to transform a sample AFP document that represents all documents with images. You then identify the image entries and define them in the configuration file.

The image information in the configuration file is used to identify the images in the AFP document and map them during the transform process. Figure 11 on page 40 shows an example of image information for different images. Each IMAGE tag along with its corresponding IMAGE_END tag defines a single image information entry in the configuration file. The first value for position and size is the horizontal dimension and the second value is the vertical dimension. The position measurements are for the upper, left-hand corner of the image relative to the upper, left-hand corner of the page.

```
<IMAGE position:(5.250in,0.613in) size:(0.667in,0.800in)>
<IMAGE_END>
<IMAGE position:(0.863in,8.483in) size:(2.400in,0.667in)>
<IMAGE_END>
<IMAGE position:(3.596in,8.550in) size:(2.633in,0.700in)>
<IMAGE_END>
<IMAGE name:(S1PSEG01) position:(6.162in,8.483in) size:(2.067in,0.604in)>
<IMAGE_END>
```

Figure 11. Image information in the image map configuration file

By default, the image map configuration file is named `imagemap.cfg`. AFP2PDF Transform looks for the file in the same directory in which the program was installed. However, you can specify a different location and name for the file.

To create the image map configuration file:

1. Create or modify the AFP2PDF Transform options file (see “AFP2PDF Transform options file” on page 10) with the entry **ImageMapEntries_File=***outputfile*, where *outputfile* is the location and name of the file for the AFP file image information. For example, `ImageMapEntries_File=c:\imagemap.out`.
2. Enter **afp2pdf** *afpfile* to run the AFP2PDF Transform, where *afpfile* is the directory and file name of the AFP document you are transforming. For example, `afp2pdf c:\documents\afpdoc.afp`.

   The system generates an output file with image information for the AFP document and a PDF file for the AFP document. For example:

   ```
   c:\imagemap.out
   c:\documents\afpdoc.pdf
   ```
3. Copy the image lines in the output file (such as `imagemap.out`) into the image map configuration file (`imagemap.cfg` by default). These image entries are "empty entries" by default. Empty entries do not include any image information between the starting <IMAGE> and ending <IMAGE_END> lines.
4. Add image information between the starting <IMAGE> and ending <IMAGE_END> lines for the images you want generated in the PDF output file.

The image information in the configuration file is used to identify the images in the AFP document. When the transform matches an image in the AFP document with image information in the configuration file, it generates the image in the PDF output file.

Identifying AFP images in the image map configuration file

In some cases, images in the configuration file can be identified from the name or the position and size information. In other cases, it might be difficult to distinguish one image from another. In these cases it is possible to work with the **afp2pdf** transform command to visually identify each of the images. By creating empty image entries and then commenting out a single entry in the file, you can identify the image when you rerun the transform and the image is generated in the output PDF file.

To identify individual images:

1. Define empty image information entries for all of the images in the image map configuration file. Empty entries do not include any image information between the starting <IMAGE> and ending <IMAGE_END> lines. For example:

```
<IMAGE position:(0.863in,8.483in) size:(2.400in,0.667in)>
<IMAGE_END>
```

2. Comment out the first image in the configuration file by adding slashes before the entry. For example:

```
//<IMAGE position:(5.250in,0.613in) size:(0.667in,0.800in)>
//<IMAGE_END>
<IMAGE position:(0.863in,8.483in) size:(2.400in,0.667in)>
<IMAGE_END>
<IMAGE position:(3.596in,8.550in) size:(2.633in,0.700in)>
<IMAGE_END>
<IMAGE name:(S1PSEG01) position:(6.162in,8.483in) size:(2.067in,0.604in)>
<IMAGE_END>
```

3. Run **afp2pdf** to generate a PDF file that contains only the image that you commented out. For example: `afp2pdf c:\documents\afpdoc.afp`, where `c:\documents\afpdoc.afp` is the directory and file name of the AFP document you are transforming.
4. View the PDF file. The image generated in the file is the image entry you commented out in the configuration file.

Removing images using the image map configuration file

The image information in the configuration file is used to identify the images in the AFP document and map them to the PDF file during the transform process. If a matching image entry in the configuration file contains an "empty entry", the image is not generated in the PDF output file. Empty entries do not include any image information between the starting <IMAGE> and ending <IMAGE_END> lines, as in Figure 12.

```
<IMAGE position:(0.863in,8.483in) size:(2.400in,0.667in)>
<!-- IMAGE_END -->
```

Figure 12. Empty entries in the image map configuration file

Therefore, to remove AFP file images so they are not generated in the PDF output file, create empty image entries in the configuration file.

To remove images:

1. Define empty image information entries in the image map configuration file for those images you do not want generated in the PDF file.
2. Run **afp2pdf** on the AFP file (for example, `afp2pdf c:\documents\afpdoc.afp`). A PDF file is generated (such as `c:\documents\afpdoc.pdf`) that does not contain any images with empty entries in the configuration file.

Substituting existing images with AFP2PDF Transform

During the AFP2PDF Transform process, you can use the image map configuration file to substitute an AFP image with a previously generated image. The only type of image which can be substituted at this point is the JPEG format.

To use an existing image, add IMAGE definition parameters between the starting <IMAGE> and ending <IMAGE_END> lines of an image information entry in the image map configuration file. The image definition parameters are:

XPos=*n*
: Defines the position, in units of 1440 units per inch, of the left edge of the image relative to the left edge of the page.

YPos=*n*
: Defines the position, in units of 1440 units per inch, of the top edge of the image relative to the top edge of the page.

XSize=*n*
: Defines the target area width (horizontal size in units of 1440 units per inch) which the image is scaled to.

YSize=*n*
: Defines the target area height (vertical size in units of 1440 units per inch) which the image is scaled to.

Filename=*path*
: Specifies the fully qualified location and file name for the image. This text should be enclosed in double quotes if a blank is used within the value.

ColorFlag=0 | 1
: Specifies the type of image to be substituted. This is an optional parameter and if specified, should be set to “1”.

As shown in Figure 13, when the first image is encountered it is substituted with “logo1.jpg”; when the second image is encountered, it is substituted with “logo2.jpg”.

```
<IMAGE position:(5.250in,0.613in) size:(0.667in,0.800in)>
IMAGE XPos=0 YPos=0 XSize=900 YSize=200 Filename="c:\images\logo1.jpg"
<IMAGE_END>
<IMAGE position:(0.863in,8.483in) size:(2.400in,0.667in)>
IMAGE XPos=0 YPos=0 XSize=500 YSize=300 Filename="c:\images\logo2.jpg"
<IMAGE_END>
<IMAGE position:(3.596in,8.550in) size:(2.633in,0.700in)>
<IMAGE_END>
<IMAGE name:(S1PSEG01) position:(6.162in,8.483in) size:(2.067in,0.604in)>
<IMAGE_END>
```

Figure 13. Example of existing images in the image map configuration file

You can define abbreviated versions of the image entries to expand the matching capabilities of incoming AFP images. This can simplify and reduce the number of image entries defined in the configuration file. To define abbreviated versions, edit the image entry and specify any combination of name, position, or size. If the incoming AFP image matches all the characteristics listed for the image entry, the image is substituted.

For example, in Figure 14, the page segment reference name, S1PSEG01, is used. When an incoming AFP image matches the name, the substituted image is added to the output. The XPos, YPos, XSize, and YSize parameters for the substituted image are extracted from the AFP image.

```
<!-- IMAGE name:(S1PSEG01) -->
IMAGE Filename="c:\images\logo1.jpg"
<!-- IMAGE_END -->
```

Figure 14. Example of abbreviated image entries in the image map configuration file

Substituting AFP shaded images with colored areas

Many AFP documents contain areas on the page that are shaded with a gray box. This is accomplished in the AFP data stream by defining an image that has pels laid out in a regular checkerboard-like pattern to create a gray shading effect. When trying to display this type of image, however, it often becomes distorted due to the scale factor and resolution of the display hardware. To avoid this problem, you can define a colored area to be used instead of the shaded image.

To substitute a shaded image with a color area, add SHADED_AREA definition parameters between the starting <IMAGE> and ending <IMAGE_END> lines of an image information entry in the image map configuration file. The shaded area definition parameters are:

XPos=*n*
: Defines the position, in inches, of the left edge of the colored area relative to the left edge of the page. This parameter is optional because the horizontal position of the incoming AFP image is used if XPos is not specified.

YPos=*n*
: Defines the position, in inches, of the top edge of the colored area relative to the top edge of the page. This parameter is optional because the vertical position of the incoming AFP image is used if YPos is not specified.

XSize=*n*
: Defines the width (horizontal size in inches) of the colored area on the page. This parameter is optional because the horizontal size of the incoming AFP image is used if XSize is not specified.

YSize=*n*
: Defines the height (vertical size in inches) of the colored area on the page. This parameter is optional because the vertical size of the incoming AFP image is used if YSize is not specified.

Shade_R=*n*
Shade_G=*n*
Shade_B=*n*
: Specifies the intensity of the red, green, and blue colors (respectively) used when generating the color of the area. The values given must be in the range 0.0 to 1.0. When all three values are set to “1.0”, white is generated. When all three values are set to “0.0”, black is generated.

: If these parameters are not specified, the transform tries to use the color described by the Shade_RGB parameter in the AFP2PDF Transform options file (see “AFP2PDF Transform options file” on page 10). By default, Shade_R=0.8, Shade_G=0.8, and Shade_B=0.8 creates a light gray color that is used if no other color specification is found.

For example, in Figure 15 on page 44 when the first image is encountered, it is substituted with a red, 0.667 x 0.8 inch area positioned at 5.25 x 0.613 inches.

```
<IMAGE position:(5.250in,0.613in) size:(0.667in,0.800in)>
SHADED_AREA XPos=5.250 YPos=0.613 XSize=0.667 YSize=0.800 Shade_R=1.0 Shade_G=0.0 Shade_B=0.0
<IMAGE_END>
<IMAGE position:(0.863in,8.483in) size:(2.400in,0.667in)>
<IMAGE_END>
<IMAGE position:(3.596in,8.550in) size:(2.633in,0.700in)>
<IMAGE_END>
<IMAGE position:(6.162in,8.483in) size:(2.067in,0.604in)>
<IMAGE_END>
```

Figure 15. Example of colored areas in the image map configuration file

You can define abbreviated versions of the image entries to expand the matching capabilities of incoming AFP images. This can simplify and reduce the number of image entries defined in the configuration file. To define abbreviated versions, edit the image entry and specify any combination of name, position, and size. If the incoming AFP image matches all the characteristics listed for the image entry, the shaded area is substituted. For example, in Figure 16, the size values are used. When an incoming AFP image matches the image size information, a colored area with the default color is created. The XPos, YPos, XSize, and YSize parameters for the rectangle are extracted from the AFP image.

```
<IMAGE size:(0.667in,0.800in)>
 SHADED_AREA
 <IMAGE_END>
```

Figure 16. Example of abbreviated colored areas in the image map configuration file

Adding an image to the transform output

Occasionally, preprinted forms are used during the printing process. These preprinted forms might have a company logo, a table, or grid that is filled in with the print data. AFP2PDF Transform can include an image, which emulates the preprinted form, in the transformed output. The transform opens the specified image file, which currently can only be JPEG format, processes the image data, and converts it into an image for the PDF output. The image can be in color and you can specify which pages it is included on.

To include an image that emulates a preprinted form, add one or more of these static image definitions between the starting <IMAGE> and ending <IMAGE_END> lines of the image information entry in the image map configuration file:

STATICIMG_PAGE
: Specifies that the same image is included on the pages specified with the Type parameter.

STATICIMG_FRONT
: Specifies that the image on the front duplex page of AFP data is placed in the PDF on pages specified with the Type parameter.

STATICIMG_BACK
: Specifies that the image on the back duplex page of AFP data is placed in the PDF on pages specified with the Type parameter.

STATIC_IMG
: Specifies that the same image is included on the pages specified with the Type parameter and all position and size dimensions are in 72 units per inch.

These parameters are used with the static image definitions:

XPos=*n*
: Defines the position, in 1440 units per inch, of the left edge of the image relative to the left edge of the page.

YPos=*n*
: Defines the position, in 1440 units per inch, of the top edge of the image relative to the top edge of the page.

XSize=*n*
: Defines the width (horizontal size in 1440 units per inch) for the target area on the page for the image.

YSize=*n*
: Defines the height (vertical size in 1440 units per inch) for the target area on the page for the image.

Filename=*path*
: Specifies the fully qualified path and name for the image. Enclose the value in double quotes if a blank is used as part of the value.

Type=All | First | Second | All=First
: Specifies the pages for which the image should be included. The values include:

  All
  : The image is included on all pages.

  First
  : The image is included on the first page only.

  Second
  : The image is included on the second page only.

  All-First
  : The image is included on all the pages except for the first page.

If the page orientation switches between portrait (length is greater than its width) and landscape (width is greater than its length), these parameters can also be specified to control the position and sizing for the landscape orientation:

XPos_Wide=*n*
: Defines the position, in 1440 units per inch, of the left edge of the image relative to the left edge of the page.

YPos_Wide=*n*
: Defines the position, in 1440 units per inch, of the top edge of the image relative to the top edge of the page.

XSize_Wide=*n*
: Defines the width (horizontal size in 1440 units per inch) for the target area on the page for the image.

YSize_Wide=*n*
: Defines the height (vertical size in 1440 units per inch) for the target area on the page for the image.

In the example in Figure 17 on page 46, the JPEG image, “form1.jpg”, is included on all simplex pages in the PDF output.

```
<IMAGE>
STATICIMG_PAGE XPOS=0 YPOS=0 XSIZE=12240 YSIZE=15840 FILENAME="C:\afp2pdf\form1.jpg" TYPE=ALL
<IMAGE_END>
```

Figure 17. Example of an image added to PDF output

Notes:

1. The image definitions do not include position or size information. You must manually add the starting and ending lines to the image map configuration file, in addition to the static image definitions.
2. You can define multiple images on a page; however, make sure the size and position do not cause the images to overlap.

Storing frequently used images with AFP2PDF Transform

In many AFP documents, the same image is used many times throughout the document, such as a company logo that appears on each page of a document. It is possible to store this image once and then reference it each time it is used. Storing frequently used images reduces the size of the resulting PDF file.

To store an image, add a CACHE_IMG definition between the starting <IMAGE> and ending <IMAGE_END> lines of an image information entry in the image map configuration file. The cache image definition has this parameter:

NAME
Defines the name of the image to be stored. The name cannot contain these words: SHADED_AREA, IMAGE, PROCESS_IMG, or CACHE_IMG. For example, NAME=IMAGE0 is not allowed. The name must be different for each image stored.

Figure 18 shows how to store frequently used images. When the first image is encountered, it is stored with the name “IMG0”.

```
<IMAGE position:(5.250in,0.613in) size:(0.667in,0.800in)>
CACHE_IMG NAME=IMG0
<IMAGE_END>
<IMAGE position:(0.863in,8.483in) size:(2.400in,0.667in)>
<IMAGE_END>
<IMAGE position:(3.596in,8.550in) size:(2.633in,0.700in)>
<IMAGE_END>
<IMAGE position:(6.162in,8.483in) size:(2.067in,0.604in)>
<IMAGE_END>
```

Figure 18. Example of stored images in image map configuration file

Chapter 6. Application programming interfaces

The AFP2PDF Transform application programming interfaces (APIs) convert AFP documents and resources into files that can be viewed with Adobe Acrobat. These APIs are written to interface with a C/C++ application.

The AFP2PDF Transform APIs are available as a dynamic link library (DLL) on the
| Windows (32-bit) server, as a shared library on the UNIX servers, and as an
| integrated file system object on an IBM i OnDemand server. Therefore, the transform function can be contained within the calling application's process and a separate process does not need to be created (as compared to using the AFP2PDF Transform command line interface).

The AFP2PDF Transform APIs use data buffers for input and output to the transform. Many times the AFP data is retrieved from a database in a buffer in memory. The pointer to this memory block is then passed directly to the conversion code for processing. The output from the API program also uses a pointer to a data buffer that contains the transformed output. Therefore, the overhead of opening, reading, and closing files is eliminated. Because system performance degrades if very large memory buffers are allocated, this approach assumes that the input and output buffers are not extremely large. The command line interface, which uses files for input and output to the transform, might need to be used for large memory buffers.

This chapter describes the packaging information for the APIs, shows examples for loading API code and obtaining function pointers, and describes the APIs for the AFP2PDF Transform.

Packaging information

| This section describes the API packaging for the AFP2PDF Transform on the
| Windows, UNIX, and IBM i OnDemand servers.

Windows server

The API package consists of one or more dynamic link library (DLL) files and a `\font` subdirectory in the directory where the DLLs are located. The `\font` subdirectory contains the font definition files needed by AFP2PDF Transform as well as several subdirectories. Figure 19 on page 48 shows the file structure for APIs used by AFP2PDF Transform.

```
afp2pdfd.dll
afp2pdfpe.dll
apf2pdfpl.dll
\java_api
    afp2pdf.jar
    a2pjni.dll
\font
    afpfont.fnt
    alias.fnt
    coded.fnt
    cpdef.fnt
    csdef.fnt
    icoded.fnt
    \AFM (optional)
    \maps
    \Type1 (optional)
```

Figure 19. File structure for AFP2PDF Transform APIs on Windows servers

UNIX server

The package consists of a shared library file (`afp2pdf_shr`) and a `/font` subdirectory in the directory where the shared library is located. The `/font` subdirectory contains the font definition files needed by AFP2PDF Transform as well as several subdirectories. The package also includes the `afp2pdfpe` and `afp2pdfpl` files, which the transform uses. Figure 20 shows the file structure for APIs used by AFP2PDF Transform.

```
afp2pdfpe
afp2pdfpe_64 (on supported 64 bit systems)
afp2pdfpl
afp2pdfpl_64 (on supported 64 bit systems)
afp2pdf_shr
afp2pdf_shr_64 (on supported 64 bit systems)
/java_api
    afp2pdf.jar
    liba2pjni.so (or .sl)
    liba2pjni_64.so (or .sl) (on supported 64 bit systems)
/font
    afpfont.fnt
    alias.fnt
    coded.fnt
    cpdef.fnt
    csdef.fnt
    icoded.fnt
    /AFM  (optional)
    /maps
    /Type1 (optional)
```

Figure 20. File structure for AFP2PDF Transform APIs on UNIX servers

IBM i OnDemand server

The package consists of a shared library file (`afp2pdf_shr`) and a `/font` subdirectory in the directory where the shared library is located. The `/font` subdirectory contains the font definition files needed by AFP2PDF Transform and several subdirectories. The package also includes the `afp2pdfpe` and `afp2pdfpl` files, which the transform uses. Figure 21 on page 49 shows the directory structure for the AFP2PDF Transform.

```
/QIBM/ProdData/OnDemand/www/binpdf
/QIBM/ProdData/OnDemand/www/binpdf/font
/QIBM/ProdData/OnDemand/www/binpdf/font/maps
/QIBM/ProdData/OnDemand/www/binpdf/java_api
/QIBM/ProdData/OnDemand/www/binpdf/locale/uconvtab
```

Figure 21. Directory structure for AFP2PDF Transform on IBM i OnDemand servers

Figure 22 shows the file structure for APIs used by AFP2PDF Transform.

```
afp2pdf
afp2pdf_shr
afp2pdfd.h
afp2pdfpe
afp2pdfpl
insure.afp
split_afp2pdf
/font
    afpfont.fnt
    alias.fnt
    coded.fnt
    cpdef.fnt
    csdef.fnt
    icoded.fnt
    /maps
/java_api
    a2peip.jar
    afp2pdf.jar
    afp2pdf.pdf
    AFP2PdfDemo.java
    AFP2PdfServlet.java
    eClient_howto.pdf
    liba2pjni.so
    testa2pd.java
/locale/uconvtab
    Ibm-1363
    Ibm-1386
    Ibm-300
    Ibm-834
    Ibm-835
    Ibm-837
    Ibm-943
    Ibm-950
```

Figure 22. File structure for AFP2PDF Transform APIs on IBM i Ondemand servers

Loading code and obtaining function pointers

This section provides Windows, UNIX, and IBM i OnDemand server examples for dynamically loading the API code and obtaining the function pointers to the API methods.

Windows server

Figure 23 on page 50 shows an API DLL dynamically loaded with the Windows LoadLibrary function.

```
// Load the Afp2pdf transform DLL.
//
HINSTANCE hXForm = LoadLibrary("c:\\afp2pdf\\afp2pdfd.dll");
if (hXForm == NULL) {
  fprintf(stderr, "Could not load the AFP2pdf transform DLL");
}
```

Figure 23. Example of loading a Windows API DLL

Figure 24 shows how to dynamically obtain the function pointer for the A2PDocStart function with the Windows GetProcAddress call and then call the A2PDocStart function with the function pointer.

```
// Define a fpA2PDocStart function pointer variable for the A2PDocStart function.
//
void* (__cdecl * fpA2PDocStart)(unsigned char*, unsigned long, char*);

// Obtain the A2PDocStart function pointer / entry point in the DLL.
//
fpA2PDocStart = (void* (__cdecl *)(unsigned char*, unsigned long, char *))
                GetProcAddress(hXForm, "A2PDocStart");
if (fpA2PDocStart == NULL) {
  fprintf(stderr, "Could not load the function pointer for A2PDocStart");
}

// Call the transform function to start processing an AFP document.
//
void* hDocument = (*fpA2PDocStart)(pInBuffer, iDataLen, "c:\\afp2pdf\\");
```

Figure 24. Example of obtaining the function pointer to the API options in Windows

UNIX server

The following sections show how to dynamically load an API shared library and obtain the function pointer on UNIX servers. On AIX, the load function returns the function pointer to the main entry point. In all other UNIX environments, the open/load function returns a handle to the shared library.

Dynamically loading an API shared library

Examples of how to dynamically load an API shared library are shown in Figure 25 for AIX, Figure 26 on page 51 for Sun and Linux, and Figure 27 on page 51 for z/OS UNIX System Services.

AIX:

```
// Declare the function pointer for the main entry point of the shared library.
//
void* (*fpA2PGetFuncPtr)(const char*);

// Load the Afp2pdf transform shared library which returns the main entry point.
//
fpA2xGetFuncPtr=(void* (*)(const char*)) load ("/a2pXForm/afp2pdf_shr", 1, NULL);
if (fpA2PGetFuncPtr == NULL) {
  fprintf(stderr, "Could not load the AFP2pdf transform library");
}
```

Figure 25. Example of loading an API shared library on AIX

Sun and Linux:

```
// Load the Afp2pdf transform shared library which returns handle to the library.
//
void* hSLib= dlopen ("/a2pXForm/afp2pdf_shr", RTLD_LAZY);
if (hSLib == NULL) {
fprintf(stderr, "Could not load the AFP2pdf transform library");
}
```

Figure 26. Example of loading an API shared library on Sun and Linux

z/OS UNIX System Services:

```
// Load the Afp2pdf transform shared library which returns handle to the library.
//
dllhandle* hSLib= dllload ("/a2pXForm/afp2pdf_shr");
if (hSLib == NULL) {
fprintf(stderr, "Could not load the AFP2pdf transform library");
}
```

Figure 27. Example of loading an API shared library on z/OS UNIX System Services

Dynamically obtaining the function pointer

Examples of how to dynamically obtain the function pointer are shown in Figure 28 for AIX, Figure 29 on page 52 for Sun and Linux, and Figure 30 on page 52 for z/OS UNIX System Services. The function pointer is obtained for the A2PDocStart function with the main entry point from the load call and then the A2PDocStart function is called with the function pointer.

AIX:

```
// Define a fpA2PDocStart function pointer variable for the A2PDocStart function.
//
void* (* fpA2PDocStart)(unsigned char*, unsigned long, char*);

// Obtain the A2PDocStart function pointer / entry point in the library.
//
fpA2PDocStart = (void* (*)(unsigned char*, unsigned long, char *))
                (*fpA2PGetFuncPtr)(hXForm, "A2PDocStart");
if (fpA2PDocStart == NULL) {
  fprintf(stderr, "Could not load the function pointer for A2PDocStart");
}

// Call the transform function to start processing an AFP document.
//
void* hDocument = (*fpA2PDocStart) (pInBuffer, iBytesRead, "/a2pXForm/");
```

Figure 28. Example of obtaining the function pointer to API options on AIX

Sun or Linux:

```
// Define a fpA2PDocStart function pointer variable for the A2PDocStart function.
//
void* (* fpA2PDocStart)(unsigned char*, unsigned long, char*);

// Obtain the A2PDocStart function pointer / entry point in the library.
//
fpA2PDocStart = (void* (*)(unsigned char*, unsigned long, char *))
                dlsym(hSLib, "A2PDocStart");
if (fpA2PDocStart == NULL) {
  fprintf(stderr, "Could not load the function pointer for A2PDocStart");
}

// Call the transform function to start processing an AFP document.
//
void* hDocument = (*fpA2PDocStart) (pInBuffer, iBytesRead, "/a2pXForm/");
```

Figure 29. Example of obtaining the function pointer to API options on Sun or Linux

z/OS UNIX System Services:

```
// Define a fpA2PDocStart function pointer variable for the A2PDocStart function.
//
void* (* fpA2PDocStart)(unsigned char*, unsigned long, char*);

// Obtain the A2PDocStart function pointer / entry point in the library.
//
fpA2PDocStart = (void* (*)(unsigned char*, unsigned long, char *))
                dllqueryfn (hSLib, "A2PDocStart");
if (fpA2PDocStart == NULL) {
  fprintf(stderr, "Could not load the function pointer for A2PDocStart");
}

// Call the transform function to start processing an AFP document.
//
void* hDocument = (*fpA2PDocStart) (pInBuffer, iBytesRead, "/a2pXForm/");
```

Figure 30. Example of obtaining the function pointer to API options on z/OS UNIX System Services

IBM i OnDemand server

Figure 31 shows how to dynamically load an API shared library on an IBM i OnDemand server.

```
// Declare the function pointer for the main entry point of the shared library.
//
void* (*fpA2WGetFuncPtr)(const char*);

// Load the Afp2pdf transform shared library which returns the main entry point.
//
fpA2WGetFuncPtr=(void* (*)(const char*))
  load ("/QIBM/UserData/OnDemand/www/binpdf/afp2pdf_shr", 1, NULL);
if (fpA2WGetFuncPtr == NULL) {
  fprintf(stderr, "Could not load the AFP2pdf transform library");
}
```

Figure 31. Example of loading an API shared library on an IBM i OnDemand server

Note: Obtaining the function pointer does not apply on IBM i.

AFP2PDF Transform API

The AFP2PDF Transform API converts AFP documents and resources into Adobe PDF files.

Input options structure

Due to the number of options that need to be specified for the transform process, a C/C++ header file containing a structure of the input options is created. The calling application should allocate the storage for this structure, update the parameter values in this structure as needed, and then pass the pointer to this structure to the appropriate API functions.

This structure is defined in the header file `afp2pdfd.h` and needs to be included by the calling application. The calling application should use the default "structure member alignment" when compiling. The input options structure is referenced with the name, `A2PCvtOpts`. The definition of this structure is shown in Figure 32

```
typedef struct
{
int iPageNumber;                   // Page number to convert
int iLastPageFlag;                 // Indicates last page being transformed
int iRotation;                     // Rotation setting (0 | 90 | 180 | 270)
int* piPdfOutLen;                  // Returned PDF output buffer length
char** ppPdfOutBuf;                // Returned PDF output buffer pointer
char szOptionsFile[SMAX_PATH+1];   // Fully qualified 'a2pxopts.cfg' file spec
char szImageMapFile[SMAX_PATH+1];  // Fully qualified 'imagemap.cfg' file spec
int iVersion;                      // Structure version number
char szFormDef[SMAX_PATH+1];       // Fully qualified form definition file
char szReserved[SMAX_PATH+1];      //
short fUserPW;                     // 1 = there is a User Password
                                   // 0 = there is no User Password
char szUserPW[SMAX_PATH+1];        // User Password
short fOwnerPW;                    // 1 = there is an Owner Password
                                   // 0 = there is no Owner Password
char szOwnerPW[SMAX_PATH+1];       // Owner Password
short fPerms;                      // 1 = there are flags in szPerms;
                                   // 0 = there are no flags in szPerms;
char szPerms[SMAX_PATH+1];         // Encryption Permissions
short sLinearize;                  // Linearize PDF (1=TRUE or 0=FALSE)
char szCreateTime[64];             // PDF Creation Time
char szSignTime[64];               // PDF Signature Time

} A2PCvtOpts;
```

Figure 32. Input options structure

The options in the A2PCvtOpts structure are:

iPageNumber
Specifies the page number that is to be transformed in the document. The first page in the document is represented by "1". If a page number that is not valid (for example, out of range) is specified, the first page in the document is converted.

iLastPageFlag
Specifies the last page when building a PDF file one page at a time from the AFP document. This option is only required when calling the A2PXFormPage function for the last page in the PDF file.

iRotation
Specifies the rotation value to use when transforming the file. Valid values are **0**, **90**, **180**, and **270**. Some AFP documents might have been formatted with a rotated orientation; therefore, you must use this option to align the text in an upright position.

piPdfOutLen
Specifies the pointer reference to a variable allocated by the calling application. The variable is filled in by the transform API and contains the length in bytes of the output PDF data. The output PDF data buffer is returned in the ppPdfOutBuf option. The calling application should check the value of the variable after the conversion process. A value of "0" indicates an error has occurred and any data in the output PDF data buffer is not valid.

ppPdfOutBuf
Specifies the address of a pointer reference variable allocated by the calling application. The variable is filled in by the transform API and contains a pointer to the output PDF data. The length of the output PDF data buffer is returned in the piPdfOutLen option. The calling application should check the value of the variable after the conversion process. A null value indicates an error has occurred.

szOptionsFile
Specifies a fully qualified path and file name string for the AFP2PDF Transform options file. This options file contains various settings to use when transforming the data. The options file is named `a2pxopts.cfg` by default. It is located in the same directory where the transform API code was installed. You can use this option to change the name and location of this file.

szImageMapFile
Specifies a fully qualified path and file name string for the image map configuration file. The image map configuration file lists images that require special processing. The image map configuration file is named `imagemap.cfg` by default. It is located in the same directory where the transform API code was installed. You can use this option to change the name and location of this file.

iVersion
Version number of the A2PCvtOpts structure. This number must be **3**.

szFormDef
Specifies a fully qualified path and file name string for a form definition file.

szReserved
Reserved.

fUserPW
Specifies whether the generated PDF is to have a user password. Values are:

0	No password
1	Password

szUserPW
Specifies the user password for the generated PDF file.

fOwnerPW
Specifies if the generated PDF is to have an owner password. Values are:

0	No password
1	Password

szOwnerPW
Specifies the owner password for the generated PDF file.

fPerms
Specifies if the generated PDF is to have restricted permissions set. Values are:

0	No restricted permissions
1	Restricted permissions

szPerms
Specifies the encryption permissions. Encrypted PDF in the transform is also tied to restricting certain functions when displayed in Adobe Acrobat. These codes are used to restrict one or more functions:

a	Add or modify text annotations and interactive form fields.
c	Modify the document contents.
p	Print the document.
s	Copy text and graphics from the document.

If any functions are disabled, a permissions password (also known as a *owner* or *master* password) must also be specified. Any user needing to override a restricted function must supply the correct permissions password.

SLinearize
Specifies that the generated PDF is to be linearized. Values are:

0	Not linearized
1	Linearized

szCreateTime
Specifies the time the generated PDF was created. The format of the time stamp is:

```
(YYYYMMDDHHmmSSOHH'mm')
```

where:
- **YYYY** is the year
- **MM** is the month
- **DD** is the day (01-31)
- **HH** is the hour (00-23)
- **mm** is the minute (00-59)
- **SS** is the second (00-59)
- **O** is the relationship of local time to Universal Time (UT), denoted by one of the characters +, -, or Z
- **HH** is the absolute value of the offset from UT in hours (00-23)
- **mm** is the absolute value of the offset from UT in minutes (00-59)

szSignTime
Specifies the time the generated PDF was signed. The format of the time stamp is:

```
(YYYYMMDDHHmmSSOHH'mm')
```

where:
- **YYYY** is the year
- **MM** is the month

- **DD** is the day (01-31)
- **HH** is the hour (00-23)
- **mm** is the minute (00-59)
- **SS** is the second (00-59)
- **O** is the relationship of local time to Universal Time (UT), denoted by one of the characters +, -, or Z
- **HH** is the absolute value of the offset from UT in hours (00-23)
- **mm** is the absolute value of the offset from UT in minutes (00-59)

Available programming functions

This section contains information about the programming functions available for the AFP2PDF Transform API.

A2PDocStart2

A2PDocStart2 initializes transform processing for an AFP document and returns a handle to the document as a void* type. This handle is required as input to all of the other functions and serves to identify the document initialized with this call. A document handle of NULL indicates that an error has occurred.

The format of A2PDocStart2 is:

```
void* A2PDocStart2 (unsigned char* pDataIn, unsigned long ulSizeIn, char* pszDir, A2PCvtOpts* pCvtOpts)
```

The options in the A2PDocStart2 function are:

pDataIn
Specifies a pointer to a memory block that contains the input AFP document. If there is an associated AFP resource group object, this must be included before the actual document.

ulSizeIn
Specifies the size in bytes of the amount of input data in the buffer.

pszDir
Specifies the directory where the AFP2PDF Transform API code is installed.

pCvtOpts
Specifies a pointer to a structure allocated by the calling program that contains the input options to be used during the transform process.

A2PGetPageCount

A2PGetPageCount returns the number of pages in a specific AFP document. The format of A2PGetPageCount is:

```
int A2PGetPageCount (void* hAfpDoc)
```

The option in the A2PGetPageCount function is:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2PDocStart function.

A value greater than "0" indicates a successful completion. A “0” value or negative number indicates an error has occurred. For example:

-10	Null value is specified for the hAfpDoc option.
-20	hAfpDoc optionr specified is not valid.

A2PXFormDoc

A2PXFormDoc transforms the entire AFP document using the given values in the input options structure. The format of A2PXFormDoc is:

```
void* A2PXFormDoc (void* hAfpDoc, A2PCvtOpts* pCvtOpts)
```

The options in the A2PXFormDoc function are:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2PDocStart function.

pCvtOpts
Specifies a pointer to a structure allocated by the calling program that contains the input options to be used during the transform process.

A value of "0" indicates a successful completion. A negative value indicates an error has occurred. For example:

-10	Null value is specified for the hAfpDoc option.
-20	hAfpDoc option specified is not valid.
-30	PDF output buffer values specified in the pCvtOpts structure are not valid.

A2PXFormPage

A2PXFormPage transforms a page in an AFP document using the given values in the input options structure. Using this function, a subset of pages from the AFP document can be used to build the PDF file.

Note: The iLastPageFlag in the input options structure must be set to "1" when calling this function for the last page in the PDF file.

The format of A2PXFormPage is:

```
void* A2PXFormPage (void* hAfpDoc, A2PCvtOpts* pCvtOpts)
```

The options in the A2PXFormPage function are:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2PDocStart function.

pCvtOpts
Specifies a pointer to a structure allocated by the calling program that contains the input options to be used during the transform process.

A value of "0" indicates a successful completion. A negative value indicates an error has occurred. For example:

-10	Null value is specified for the hAfpDoc option.
-20	hAfpDoc option specified is not valid.
-30	PDF output buffer values specified in the pCvtOpts structure are not valid.

A2PDocEnd

A2PDocEnd ends the processing for a given AFP document. The format of A2PDocEnd is:

```
void* A2PDocEnd (void* hAfpDoc, A2PCvtOpts* pCvtOpts)
```

The options in the A2PDocEnd function are:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2PDocStart function.

pCvtOpts
Specifies a pointer to a structure allocated by the calling program that contains the input options to be used during the transform process.

A value of "0" indicates a successful completion. A negative value indicates an error has occurred. For example:

-10	Null value is specified for the hAfpDoc option.
-20	hAfpDoc option specified is not valid.

A2PGenerateMessages

A2PGenerateMessages allows the suppression of warning and error messages generated by the transform while converting a document. If this function is not called, the transform generates messages by default.

The format of A2PGenerateMessages is:

```
int A2PGenerateMessages (void* hAfpDoc, int iSwitch)
```

The parameters in the A2PGenerateMessages function are:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2PDocStart function.

iSwitch
Specifies the whether messages should be generated. A value or “0” suppresses all messages for the document. All nonzero values generate messages.

A value of “0” indicates a successful completion. A negative value indicates an error has occurred and the function did not complete. For example:

-10	Null value is specified for the hAfpDoc option.

A2PBufferMessages

A2PBufferMessages allows the buffering of messages for a document so the message text can then be retrieved with the A2PGetMessageBuffer function.

The format of A2PBufferMessages is:

```
int A2PBufferMessages(void* hAfpDoc, int iSwitch);
```

The options in the A2PBufferMessages function are:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2PDocStart function.

iSwitch
Specifies whether the messages should be buffered. A value of “0” suppresses the buffering of messages for the document. All nonzero values turn on message buffering.

A2PGetMessageBuffer

A2PGetMessageBuffer returns the message buffer. The format of A2PGetMessageBuffer is:

```
char* A2PGetMessageBuffer(void* hAfpDoc)
```

The option in the A2PGetMessageBuffer function is:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2PDocStart function.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at `www.ibm.com/legal/copytrade.shtml`

Adobe, Acrobat, Portable Document Format (PDF), and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Portions of the OnDemand Windows client program contain licensed software from Pixel Translations Incorporated, © Pixel Translations Incorporated 1990, 2003. All rights reserved.

Other company, product or service names may be trademarks or service marks of others.

Glossary

This glossary defines technical terms and abbreviations used in this publication.

These cross-references are used in this glossary:

- **See.** Refers to preferred synonyms or to defined terms for acronyms and abbreviations.
- **See also.** Refers to related terms that have similar, but not synonymous, meanings, or to contrasted terms that have opposite or substantively different meanings.

A

ACIF. See AFP Conversion and Indexing Facility.

Adobe Acrobat. An Adobe Systems, Incorporated software product that lets you create and share PDF documents and forms.

Adobe Font Metrics (AFM). An Adobe Systems, Incorporated file containing PostScript font information that is used to format documents for PostScript devices.

Adobe Reader. An Adobe Systems, Incorporated software product that views and interacts with PDF files.

Advanced Function Presentation (AFP). A set of licensed programs, together with user applications, that use the all-points-addressable concept to print data on a wide variety of printers or to display data on a variety of display devices. AFP includes creating, formatting, archiving, retrieving, viewing, distributing, and printing information.

Advanced Interactive Executive (AIX). A UNIX operating system developed by IBM that is designed and optimized to run on POWER microprocessor-based hardware, such as servers, workstations, and blades.

AFM. See Adobe Font Metrics.

AFP. See Advanced Function Presentation.

AFP Conversion and Indexing Facility (ACIF). A software product that converts a print file into a MO:DCA-P document, creates an index file for later retrieval and viewing, and retrieves resources used by an AFP document into a separate file.

AIX. See Advanced Interactive Executive.

American Standard Code for Information Interchange (ASCII). A standard code used for information exchange among data processing systems, data communication systems, and associated equipment. ASCII uses a coded character set consisting of 7-bit coded characters. See also Extended Binary Coded Decimal Interchange Code.

API. See application programming interface.

application programming interface (API). An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

ASCII. See American Standard Code for Information Interchange.

B

band. An arbitrary layer of an image. An image can consist of one or more bands of data.

bi-level image. An image in which each picture element is represented by only one bit; therefore, a bi-level image contains only two colors: a background color and a foreground color.

C

case-sensitive. Pertaining to the ability to distinguish between uppercase and lowercase letters.

character identifier. The standard identifier for a character, regardless of its style. For example, all uppercase A's have the same character identifier.

character set. A defined set of characters that can be recognized by a configured hardware or software system. See also font character set.

CMYK. A color model used by the printing industry based on mixing cyan, magenta, yellow, and black.

coded font. In AFP support, a font file that associates a code page and a font character set. For double-byte fonts, a coded font associates multiple pairs of code pages and font character sets.

code page. A particular assignment of code points to graphic characters. Within a given code page, a code point can only represent one character. A code page also identifies how undefined code points are handled. See also coded font.

code page global identifier (CPGID). A 5-digit decimal or 2-byte binary identifier that is assigned to a code page. The range of values is 00001 to 65534 (X'0001' to X'FFFE').

Content Manager OnDemand. An IBM program that lets you automatically capture, index, archive, search, retrieve, present, and reproduce stored computer-generated documents and other business-related data.

CPGID. See code page global identifier.

D

DBCS. See double-byte character set.

default. Pertaining to an attribute, value, or option that is assumed when none is explicitly specified.

deprecated. Pertaining to an entity, such as a programming element or feature, that is supported but no longer recommended and that might become obsolete.

double-byte character set (DBSC). A set of characters in which each character is represented by two bytes. These character sets are commonly used by national languages, such as Japanese and Chinese, that have more symbols than can be represented by a single byte.

duplex. Pertaining to printing on both sides of a sheet of paper. See also simplex.

E

EBCDIC. See Extended Binary Coded Decimal Interchange Code.

Extended Binary Coded Decimal Interchange Code (EBCDIC). A coded character set of 256 eight-bit characters developed for the representation of textual data. EBCDIC is not compatible with ASCII character coding. See also American Standard Code for Information Interchange.

F

FGID. See font typeface global identifier.

file. A collection of related data that is stored and retrieved by an assigned name. A file can include information that starts a program (program-file object), contains text or graphics (data-file object), or processes a series of commands (batch file).

Flate compression. A zlib/deflate compression method that uses a lossless algorithm to compress PDF files.

font. (1) A family or assortment of characters of a given size and style, for example, 9-point Bodoni modern. A font has a unique name and might have a registry number. (2) A particular type style (for example, Bodoni or Times Roman) that contains definitions of character sets, marker sets, and pattern sets. See also coded font.

font character set. Part of an AFP font that contains the raster patterns, identifiers, and descriptions of characters. See also character set.

font mapping. A file or table that compares and matches one type of font to another; for example, core raster fonts to core outline fonts.

font metrics. Measurement information that defines individual character values, such as height, width, and space, as well as overall font values, such as averages and maximums. Font metrics can be expressed in specified fixed units, such as pels, or in relative units that are independent of both the resolution and size of the font.

font typeface global identifier (FGID). A unique font identifier that can be expressed as either a 2-byte binary value or a 5-digit decimal value. The FGID is used to identify a type style and these characteristics: posture, weight, and width.

form definition. An AFP resource object that defines the characteristics of the form or printed media, including: overlays to be used, duplex printing, text suppression, the position of composed-text data on the form, and the number and modifications of a page.

G

GOCA. See Graphics Object Content Architecture.

Graphics Object Content Architecture (GOCA). An architecture that provides a collection of graphics values and control structures used to interchange and present graphics data.

H

hexadecimal. Pertaining to a numbering system with base of 16.

I

image. (1) A pattern of toned and untoned pels that form a picture. (2) An electronic representation of an original document or picture produced by a scanning device or created from software.

image data. (1) A pattern of bits with 0 and 1 values that define the pels in an image. A 1-bit is a toned pel. (2) Digital data derived from electrical signals that represent a visual image. (3) Rectangular arrays of raster information that define an image.

indexing. A process of matching reference points within a file and creating structured field tags within the AFP document and the separate index object file.

| **integrated file system.** A function of the IBM i operating system that supports stream input/output and storage management in a manner that is similar to personal computer and UNIX operating systems, while providing an integrating structure over all information stored on a system.

inline resource. A resource contained in a file with the AFP document data.

L

library. (1) A system object that serves as a directory to other objects. A library groups related objects, and allows the user to find objects by name. (2) A data file that contains copies of a number of individual files and control information that allows them to be accessed individually.

Linux. An open source operating system that runs on a wide range of hardware platforms and has many features that are similar to the UNIX system.

M

mapping. (1) The process of transforming data, including images, from one format to another. See also font mapping. (2)

N

null value. A parameter position for which no value is specified.

N_up. The partitioning of a side of a sheet into a fixed number of equal size partitions. For example, N_up 4 divides each side of the sheet into four equal partitions.

O

object. In AFP architecture, a collection of structured fields, bounded by a begin-object function and an end-object function. The object can contain other structured fields containing data elements of a particular type. Examples of objects are text, fonts, graphics and images.

OpenType font. An extension of the TrueType font format that adds support for PostScript outlines and more support for international character sets and advanced typographic control.

outline font. A font whose graphic character shapes are defined by mathematical equations rather than by raster patterns. See also raster font.

overlay. A resource object that contains presentation data, such as text, image, graphics, and bar code data. Overlays define their own environment and are often used as electronic forms.

P

page segment. An AFP resource object containing text, image, graphics, or bar code data that can be positioned on any addressable point on a page or an electronic overlay.

parameter. A value or reference passed to a function, command, or program that serves as input or to control actions. The value is supplied by a user or by another program or process.

PDF. See Portable Document Format.

pel. See picture element.

PFB. See Printer Font Binary.

PFM. See Printer Font Metrics.

picture element (pel). (1) An element of a raster pattern about which a toned area on the photoconductor might appear. (2) The smallest printable or displayable unit that can be displayed. A common measurement of device resolution is picture elements per inch.

Portable Document Format (PDF). A standard specified by Adobe Systems, Incorporated, for the electronic distribution of documents. PDF files are compact; can be distributed globally through e-mail, the Web, intranets, or CD-ROM; and can be viewed with the Acrobat Reader.

PostScript. A page description language developed by Adobe Systems, Incorporated that describes how text and graphics are presented on printers and display devices.

Printer Font Binary (PFB). A file used for installing fonts in a Windows system. The font file must be unpacked into an ASCII form for downloading to PostScript-language printers.

Printer Font Metrics (PFM). A binary font metrics file containing information approximately equivalent to that in an AFM file. The PFM file includes character widths, the Windows font menu name, kerning information, and a flag indicating whether printer drivers should re-encode the font for printing.

R

raster font. A font in which the characters are defined directly by the raster bit map. See also outline font.

resource. A collection of instructions used, in addition to the document data, to produce the presentation output. Resources include coded fonts, font character sets, code pages, page segments, overlays, form definitions, and page definitions.

RGB. Pertaining to a color display that accepts signals representing red, green, and blue.

S

simplex. Pertaining to printing on only one side of the paper. See also duplex.

structured field. (1) A self-identifying string of bytes and its data or parameters. (2) A mechanism that permits variable length data to be encoded for transmission in the data stream.

T

TrueType font. A font format based on scalable outline technology in which the graphic character shapes are based on quadratic curves. The font is described with a set of tables contained in a TrueType font file.

U

Unicode. A character encoding standard that supports the interchange, processing, and display of text that is written in the common languages around the world, plus some classical and historical texts. For example, the text name for $ is "dollar sign" and its numeric value is X'0024'. The Unicode standard has a 16-bit character set defined by ISO 10646.

UNIX. A highly portable operating system that features multiprogramming in a multiuser environment. The UNIX operating system was originally developed for use on minicomputers, but has been adapted for mainframes and microcomputers.

W

Windows. Pertaining to a Microsoft Corporation operating system program that provides a graphical user interface for DOS.

Z

z/OS UNIX System Services. An element of z/OS that creates a UNIX environment which conforms to the XPG4 UNIX 1995 specifications and provides two open systems interfaces on the z/OS operating system: an application program interface (API) and an interactive shell interface.

Index

Program Number: 5697-N93
5724–J33
5770-RD1

SC19-2944-00