Content Manager OnDemand

AFP2HTML Transform User's Guide

SC19-2945-00

Content Manager OnDemand

AFP2HTML Transform User's Guide

SC19-2945-00

This edition replaces SC19-1288-01.

© Copyright IBM Corporation 2007, 2010.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures v

Tables vii

About this publication ix
Understanding syntax notation ix
Related information x

Summary of Changes xi

Chapter 1. Overview of AFP2HTML Transform 1
Benefits 1
Limitations 1

Chapter 2. Installing AFP2HTML Transform 3
Installing AFP2HTML Transform on Windows . . . 3
Installing AFP2HTML Transform on AIX, HP-UX, Solaris, and Linux 3
Installing AFP2HTML Transform on an IBM i OnDemand server 3
Installing AFP2HTML Transform with OnDemand. . 4
Removing AFP2HTML Transform 4

Chapter 3. Configuration 5
AFP2HTML Transform command 5
Syntax 5
Parameters 5
Return codes 7
AFP2HTML Transform options file 7
AFP2HTML Transform profile 8
Using AFP resources. 9

Chapter 4. Mapping fonts 11
Files supplied for mapping fonts 11
Coded Font File 12
Character set definition file 12
Code page definition file 15
Code page map file. 16
Alias file 17
ISO 8859-1 (Latin 1) Character Set 17
Font map configuration file 17
Process for mapping fonts 18

Chapter 5. Mapping AFP images 19
Creating the image map configuration file 19
Identifying AFP images in the image map configuration file 21
Removing images using the image map configuration file 21
Substituting existing images with AFP2HTML Transform 22
Substituting AFP shaded images with colored areas 23
Adding an image to the transform output 24

Chapter 6. Application programming interfaces 27
Packaging information. 27
Windows server 27
UNIX server 28
IBM i OnDemand server 28
Loading code and obtaining function pointers . . . 29
Windows server 29
UNIX server 30
IBM i OnDemand server 32
AFP2HTML Transform API 32
Input options structure 32
Available programming functions 35

Notices 43
Trademarks 45

Glossary 47

Index 51

© Copyright IBM Corp. 2007, 2010

Figures

1. Example of a coded.fnt file 12
2. CHARSET section of csdef.fnt file 13
3. FGID section of csdef.fnt file 14
4. CODEPG section of cpdef.fnt file 15
5. Code page map file example 16
6. Alias file example 17
7. Image information in the image map configuration file 20
8. Empty entries in the image map configuration file 21
9. Example of existing images in the image map configuration file 23
10. Example of abbreviated image entries in the image map configuration file 23
11. Example of colored areas in the image map configuration file 24
12. Example of abbreviated colored areas in the image map configuration file 24
13. Example of an image added to HTML output 25
14. File structure for AFP2HTML Transform APIs on Windows servers 28
15. File structure for AFP2HTML Transform APIs on UNIX servers 28
16. Directory structure for AFP2HTML Transform on IBM i OnDemand servers 28
17. File structure for AFP2HTML Transform APIs on IBM i OnDemand servers 29
18. Example of loading a Windows API DLL 29
19. Example of obtaining the function pointer to the API options in Windows 30
20. Example of loading an API shared library on AIX 30
21. Example of loading an API shared library on Sun and Linux 30
22. Example of loading an API shared library on z/OS UNIX System Services 31
23. Example of obtaining the function pointer to API options on AIX 31
24. Example of obtaining the function pointer to API options on Sun or Linux 31
25. Example of obtaining the function pointer to API options on z/OS UNIX System Services . 32
26. Example of loading an API shared library on an IBM i OnDemand server 32
27. Input options structure 33

© Copyright IBM Corp. 2007, 2010

Tables

1. Font files and subdirectories 11
2. Coded font files 12
3. Attribute values for CHARSET 13
4. Attribute Values for FGID. 14
5. CODEPG attributes 15

© Copyright IBM Corp. 2007, 2010

About this publication

This publication provides information about using AFP2HTML Transform. This publication helps you:

- Plan for transforming data from Advanced Function Presentation (AFP) format to Hypertext Markup Language (HTML) for viewing with a Web browser.
- Install and configure AFP2HTML Transform.
- Map fonts and images.
- Use application programming interfaces (APIs).

The information in this publication is for system programmers who install and configure AFP2HTML Transform, and for operators who use AFP2HTML Transform. This publication assumes that you are experienced using Microsoft® Windows®, UNIX®, or IBM® i systems, or Content Manager OnDemand applications.

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.

© Copyright IBM Corp. 2007, 2010

Related information

For information about the AFP2PDF Transform, see *Data Transforms AFP2PDF Transform User's Guide*, SC19-1287.

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

Summary of Changes

This publication contains additions and changes to information previously presented in *AFP2HTML Transform User's Guide*, SC19–1288–01. The technical additions and changes are marked with a revision bar (|) in the left margin.

This change has been made throughout the publication:

- References to the "IBM i" operating system have been added.

This information is new or updated:

- IBM i Common Server has been added to the server requirements in Chapter 2, "Installing AFP2HTML Transform," on page 3.
- IBM i Portable Application Solutions Environment (PASE) has been added to the UNIX environments in "Installing AFP2HTML Transform on AIX, HP-UX, Solaris, and Linux" on page 3.
- A new section, "Installing AFP2HTML Transform on an IBM i OnDemand server" on page 3, has been added.
- The documentation for "Installing AFP2HTML Transform with OnDemand" on page 4 has been updated.
- The types of AFP2HTML Transform APIs have been updated in Chapter 6, "Application programming interfaces," on page 27.
- A new section that describes API packaging information has been added for "IBM i OnDemand server" on page 28.
- A new section that describes dynamically loading the API code has been added for "IBM i OnDemand server" on page 32.
- The definition of "integrated file system" has been added to the "Glossary" on page 47.

© Copyright IBM Corp. 2007, 2010

Chapter 1. Overview of AFP2HTML Transform

AFP2HTML Transform converts Advanced Function Presentation (AFP) documents into Hypertext Markup Language (HTML) files for viewing with a Web browser.

AFP2HTML Transform lets you:

- Operate on multiple operating systems, including AIX®, HP-UX, IBM i, Linux®, Sun Solaris, Windows, and z/OS®.
- Use a C/C++ interface to integrate with applications distributing information on the Internet.
- Fully integrate with the IBM Content Manager OnDemand Web Enablement Kit and the IBM Enterprise Information Portal (EIP).
- Use configuration files to customize how AFP documents are transformed.

Benefits

With AFP2HTML Transform, you can:

- Avoid the time, disruption and expense associated with configuring client workstations. AFP2HTML Transform runs on your Web server or other back-end application server and does not require special hardware or expensive software.
- Quickly transform text and images and display documents on a Web browser with the same fidelity as if they were printed.
- Print AFP2HTML Transform documents on any local printer using the print function in the Web browser.
- Quickly retrieve your information using the Web browser's search features.

Limitations

The current limitations with the AFP2HTML Transform include:

- It is not possible to represent rotated text in HTML and Web browsers. On the Windows operating system, if any rotated text is encountered during the conversion process, the transform converts the text to image data. This text is included as part of the Graphical Image Format (GIF) output file if image data was requested. On UNIX and IBM i operating systems, any rotated text is ignored by the transform.
- On the Windows operating system, if the AFP data contains Graphics Object Content Architecture (GOCA) objects, the transform converts the graphics to image data. This data is included as part of the GIF output file if image data was requested. On UNIX and IBM i operating systems, any GOCA data is ignored by the transform.
- Bar Code Object Content Architecture (BCOCA) data in an AFP document is not supported; it is ignored by the transform.
- Using TrueType and OpenType fonts or Unicode text is not supported.
- No support is available for any object containers.

© Copyright IBM Corp. 2007, 2010

Chapter 2. Installing AFP2HTML Transform

AFP2HTML Transform is usually installed on a workstation or system running an HTTP server or an application server. These are the server and client requirements for AFP2HTML 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
  - Netscape 7.01 or later
  - Microsoft Internet Explorer 6 or later

If you plan to use the transform in conjunction with the IBM Content Manager OnDemand Web Enablement Kit, install the AFP2HTML Transform on the same workstation or server.

Installing AFP2HTML Transform on Windows

To install AFP2HTML Transform on Windows, run the `afp2web.exe` file. By default, all files are installed to the `C:\Program Files\IBM\AFP2web` directory.

Installing AFP2HTML Transform on AIX, HP-UX, Solaris, and Linux

To install AFP2HTML 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 AFP2HTML Transform on an IBM i OnDemand server

For an IBM i server with Content Manager OnDemand, the AFP2HTML Transform is delivered in two save files, QRLMINSA2H (installation code) and QRLMA2H (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 QRLMINSA2H save file:

   ```
   RSTOBJ OBJ(QRLMINSA2H) SAVLIB(QRDARS) DEV(*SAVF) SAVF(QRDARS/QRLMINSA2H) RSTLIB(QRDARS)
   ```

3. Run this program call:

   ```
   CALL QRDARS/QRLMINSA2H
   ```

Notes:

1. Symbolic links to the directory object in `/QIBM/ProdData/OnDemand/www/binhtml` have been created in `/QIBM/UserData/OnDemand/www/binhtml`.

© Copyright IBM Corp. 2007, 2010

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/binhtml` directory, which might get replaced when upgrading to a new release or installing PTF updates.

Installing AFP2HTML Transform with OnDemand

To use AFP2HTML 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 AFP2HTML Transform when processing an AFP file. At a minimum, you must make these configuration changes:

- The AFPViewing option must be “html” in the browser sections.
- The InstallDir option in the afp2html section must point to the directory on the server that contains the AFP2HTML Transform.
- To install AFP2HTML Transform, the OnDemand Internet Connection must be version 2.2.1.6 or later.

Removing AFP2HTML Transform

On Windows, to remove AFP2HTML Transform, select **Start -> Control Panel -> Add or Remove Programs**.

On AIX, HP-UX, Solaris, and Linux, to remove AFP2HTML Transform, run the *Content Manager OnDemand server directory*`/afp2web/_uninst850afp2web/uninstallafp2web` file.

Chapter 3. Configuration

This chapter describes the AFP2HTML Transform commands and files used during the configuration process. It also describes AFP resources that the transform uses.

AFP2HTML Transform command

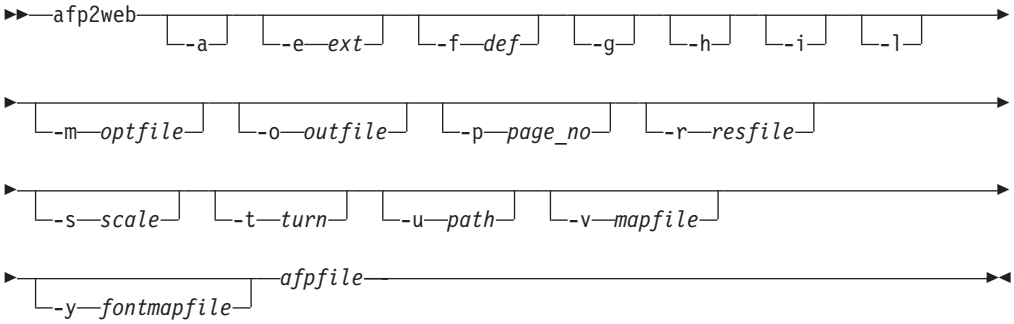
The **afp2web** command transforms AFP files and resources into files that you can distribute over the Internet and view with a Web browser.

The **afp2web** command generates files that can be used directly with a Web browser and tries to replicate the AFP printed output. Images and graphics in the AFP file can be ignored, dynamically transformed into a Graphical Image Format (GIF) image, or substituted with an existing Web browser image file, either GIF or Joint Photographic Experts Group (JPEG).

Syntax

Only a portion of the parameters needed to control the AFP2HTML Transform are available with the command. The other parameters are specified in an options file. The options file must reside in the same directory as the program module. See "AFP2HTML Transform options file" on page 7 for more information about the options file.

The syntax for the **afp2web** command is:



Parameters

-a Transforms all of the pages in the AFP input file into a separate output file for each page (except when the -h parameter is also specified). Each output file has a number concatenated to the end of the file name to distinguish it from other transformed files. For example, the transformed output file names for a three-page document titled `afpfile.afp` would be: `afpfile1.html`, `afpfile2.html`, and `afpfile3.html`. This parameter also creates an HTML file that defines a toolbar (created with JavaScript™) and a page display area. The toolbar contains buttons to navigate through pages in the document by controlling the HTML displayed in the page display area.

-e *ext*
Specifies the file extension for the HTML output, such as "htm".

© Copyright IBM Corp. 2007, 2010

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

```
-f c:\mydirectory\myformdef.fde
```

-g Generates an associated GIF file for all of the images and graphics on an HTML page. Graphic objects are supported on the Windows server only.

If all the pages for an AFP file are being transformed using the -a parameter, a separate GIF image file is generated for each AFP page, if necessary. The GIF file is generated in the same directory as the HTML output and is called out by that HTML file.

If the image map configuration file is used (for example, `imagemap.cfg`), images identified in the configuration file are not included as part of any generated image file. See Chapter 5, "Mapping AFP images," on page 19 for more information about identifying images using this configuration file.

-h Generates a single HTML output file for all of the pages in the AFP document instead of a separate file for each AFP page. You should use this parameter for transforming documents that are one to four pages in length. The performance (response time) of the Web browsers is drastically reduced if the displayed HTML file is large.

If the -g parameter is specified, a separate GIF image is generated for each AFP page, if necessary. All of the associated GIF image files are called from the HTML output file.

-i Suppresses the generation of text using fonts listed in the font configuration file (for example, `imagfont.cfg`). For more information about identifying fonts with files, see "Files supplied for mapping fonts" on page 11.

-l Turns off all of the generated error and informational console messages and sends them to the `afp2web.err` log file.

-m *optfile*
Specifies the location and name of the transform options file. This file name should not use relative paths and should be fully qualified. See "AFP2HTML Transform options file" on page 7 for more information about the options file.

-o *outfile*
Specifies the location and file name of the output HTML file. By default, the output HTML file is placed in the same directory as the input file, is given the same name as the input file, and has a file extension of ".html". For example, when the HTML is generated from an AFP file named `afpdoc.afp`, a file named `afpdoc.html` is created.

When combined with the -a parameter, the location information is used for all of the generated output files. For example, if `-o c:\html_out\abcd.html` was specified for a three-page AFP file, the three files, `abcd1.html`, `abcd2.html`, and `abcd3.html`, are generated in the `d:\html_out` directory.

-p *page_no*
Specifies the page number that is to be transformed in the AFP document. Make sure that the specified page number does not fall outside the page range of the document. If a page number is given that is not valid, the first page in the document is transformed.

The -a parameter is ignored when specified with this parameter.

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

```
-r c:\mydirectory\afpresfile.res
```

-s *scale*
Specifies the factor that scales the output. The default value is **1.0**. Specifying a value of `-s 2.0` scales the output to twice as large as the default value; specifying a value of `-s 0.50` scales the output to one half of the default value.

-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.

For Windows servers only: If the -g parameter is used, any text not in the upright position is included as part of the associated GIF image file.

For UNIX servers only: Any text not in the upright position is ignored.

-u *path*
Specifies the directory where the trace log is located.

-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 19 for more information about the image map configuration file.

-y *fontmapfile*
Specifies the location and name of the font map configuration file that is different from the default, `imagfont.cfg`. This file name should not use relative paths and should be fully qualified. See "Font map configuration file" on page 17 for more information about the font map configuration file.

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

Return codes

When the **afp2web** command runs, you see one of these return code values:

0 Successful completion of the transform.

Nonzero
An error has occurred.

AFP2HTML Transform options file

Parameters to control settings for the AFP2HTML Transform are specified in an options file. The location and name of this file is specified with the -m parameter.

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 are case-sensitive. The parameters for the AFP2HTML Transform options are:

AutoRotate
Indicates that 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. If any AFP text is formatted in a rotated orientation, the text is ignored because the function to display rotated text within HTML is not possible. Specifying this parameter 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 AutoRotate parameter overrides this setting.

FontsInPx
Indicates that the transform changes the units used to specify the font size from points (pt) to pixels (px).

LinePositioning
Indicates that the transform generates entire text strings within the HTML. By default the transform places individual words when converting the AFP text into HTML, which typically provides the best output. However, if the size of the HTML object being generated is a concern, specifying this parameter might provide HTML output that is smaller in size.

PageBordersOff
Indicates that the red box for the page outline is removed. By default the transform places a red box around the logical page dimensions described within the AFP file.

AFP2HTML Transform profile

Parameters to control certain settings for the AFP2HTML Transform are specified in a profile. By default, the name of this file is `afp2web.ini`.

When using the **afp2web** command, the `afp2web.ini` file must reside in the same directory as the program module. If running the transform function from the API, you can locate the profile in any directory on the system and you can use any file name. By setting the szProfile option in the structure passed to the transform function, different profiles can be specified for different types of documents.

The first line of the profile must include the entry **[Preferences]**. After the first line, profile parameters can be specified in any order. The parameters and their values are case-sensitive.

Parameters for the AFP2HTML Transform profile are:

FontDataPath=*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: `FontDataPath=c:\fontfils`, the code page map files must reside in the `c:\fontfils\maps` directory.

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.

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.

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" for more information.

Using AFP resources

The AFP resources used by AFP2HTML Transform include:

- Page segments
- Overlays
- Form definitions

Currently, AFP2HTML Transform does not process AFP font files. If the program encounters resources of this type, they are ignored. To display text, the transform uses the fonts available in the Web browser when converting to HTML. Font definition files that map the standard AFP fonts to the Web browser fonts 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 11.

The page segment, overlay, and form definition 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 a separate file.

When using the **afp2web** command, this resource file is specified with the -r parameter. See "Parameters" on page 5 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 AFP2HTML Transform profile. See "AFP2HTML Transform profile" on page 8 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 profile, the given file extension is used when searching for the resource. For example, if `PageSegExt=*.PSG` is set in the profile, 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 transform profile
4. The resource subdirectory where the transform modules were installed

Chapter 4. Mapping fonts

The AFP2HTML Transform must map the AFP fonts your document was created with to fonts that can be displayed with your Web browser. The AFP2HTML Transform uses font definition files that are loaded into the `\font` subdirectory during installation. The font definition files map these fonts: Core Interchange (Latin only), compatibility, coordinated, Sonoran, and Data1.

This chapter describes font definition files, the font map configuration file, and the process for mapping fonts.

Files supplied for mapping fonts

Table 1 lists the AFP2HTML Transform font support files and the subdirectories in which they are installed. (The directory is that in which the AFP2HTML Transform was installed.)

Table 1. Font files and subdirectories

File	File name	Subdirectory	Description
Coded font file	icoded.fnt, coded.fnt	\font	Specifies which AFP code page and AFP font character set make up the coded font. `coded.fnt` is optional and is meant to contain user-defined coded fonts.
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.
Code page definition file	cpdef.fnt	\font	Maps each AFP code page to a Web page browser character set and indicates which code page map file to use for the AFP2HTML 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 representing a Web browser character set.
Alias file	alias.fnt	\font	Maps the font type families to an associated font metric file; also maps the font family type to be used during the transform.
Font metric information file	*font*.AFM	\font\AFM	Contains the font metric information, which is the dimension of each of the characters.

© Copyright IBM Corp. 2007, 2010

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 AFP2HTML 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, AFP2HTML 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 on page 13 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 14 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" in Windows.
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 to a Web browser character set. 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 16 for more information about mapping code pages). The second attribute is the Web browser character set that you decide is the best match for your AFP code page. 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
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
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
wincp	ANSI or SYMBOL	ANSI	Windows character set

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

AFP2HTML 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 a Windows ANSI or SYMBOL character set.

Figure 5 shows an example of the contents of the code page map file "395.cp" for the "T1000395" code page mapped to the Windows ANSI character set.

```
;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 Windows 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 Windows 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 Windows code page contains a character that does not exist in the AFP code page. The code point for a Windows character not found in the AFP code page can be used for replacing NOMATCH characters.

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 AFP2HTML Transform to change all requests for the SonoranSerif font (which might not exist in the Web browser) to requests for the Times New Roman font (which is one of the core fonts shipped with the AFP2HTML Transform).

Note: Font family name remapping, especially to TrueType fonts, can cause misalignment of text characters because the display font is not the same as the font used to create the AFP document. Remapping of one font family name to a different font family name with very different characteristics (such as STYLE) might mean a matching font cannot be found. You receive an error message if either font substitute cannot be found.

```
[FONT]
; ***** Requested font = Type 1 font, TrueType font *****
Book=TNR,Times New Roman
CourierOverstrike=Cou,Courier New
SonoranSerif=TNR,Times New Roman
SonoranSansSerif=TNR,Arial
Text=Cou,Courier New
```

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”.

ISO 8859-1 (Latin 1) Character Set

ISO8859-1 is a standard Latin1 character set encoding that AFP2HTML Transform places in HTML header output. For more information, see CPGID 819 on this Web page: http://publib.boulder.ibm.com/printer/info/afpdb/codepage.htm

Font map configuration file

On the Windows operating system, it is possible to list one or more FGID numbers in the font map configuration file so any text for a font is converted to image data. By default, the name of this file is `imagfont.cfg`.

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

The parameter for the font map configuration file is:

fgid[**,** *fgid*][**.** *fgid*...]
Indicates one or more font typeface global ientifier (FGID) numbers.

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 AFP2HTML 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 in case the modified copy becomes corrupted:

   ```
   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 16 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 Web browser character set name 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 FGID section of the file if you are naming a new font typeface global identifier (FGID), and, optionally, identify FGIDs in the font map configuration file, `imagfont.cfg`.
6. If you have created a coded font, create or edit the `coded.fnt` file and add your coded font.

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, AFP2HTML Transform creates image information entries as comments in the HTML file. The HTML comments 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 HTML output, such as GIF, JPEG, Portable Network Graphics (PNG), or other 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 HTML display that models the use of a preprinted form used during printing.

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 transformed GIF file.

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 7 on page 20 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 line contains the page segment resource name (if available), the position value in inches and pixels, and the size value in inches and pixels. 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.

© Copyright IBM Corp. 2007, 2010

```
<!-- IMAGE position:(5.250in,0.613in) / (567px,66px) size:(0.667in,0.800in / (72px,86px) -->
<!-- IMAGE_END -->
<!-- IMAGE position:(0.863in,8.483in) / (93px,916px) size:(2.400in,0.667in) / (259px,72px) -->
<!-- IMAGE_END -->
<!-- IMAGE position:(3.596in,8.550in) / (388px,923px) size:(2.633in,0.700in) / (284px,76px) -->
<!-- IMAGE_END -->
<!-- IMAGE name:(S1PSEG01) position:(6.162in,8.483in) / (666px,916px) size:(2.067in,0.604in) / (223px,65px) -->
<!-- IMAGE_END -->
```

Figure 7. Image information in the image map configuration file

By default, the image map configuration file is named `imagemap.cfg`. AFP2HTML 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.

Note: The file examples in the following procedure 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 could be `/tmp/afpdoc.afp` in a UNIX environment.

To create the image map configuration file:

1. Enter **afp2web -a -g** *afpfile* to run the AFP2HTML Transform, where -a creates separate files for each page, -g creates GIF image files, and *afpfile* is the directory and file name of the AFP document you are transforming, such as:

   `c:\documents\afpdoc.afp`

   The transform generates these files:

   - An HTML file for page *x* of the AFP document, such as:

     `c:\documents\afpdoc`*x*`.html`
   - An associated image file for page x of the AFP document, such as:

     `c:\documents\afdoc`*x*`.gif`

   For example, the `c:\documents\afpdoc1.gif` file indicates that there is image data on the first page of the document and the `c:\documents\afpdoc1.html` file contains HTML comments for each of the images on that page, such as:

   ```
   <!-- IMAGE position:(5.250in,0.613in) / (567px,66px) size:(0.667in,0.800in / (72px,86px) -->
   <!-- IMAGE_END -->
   <!-- IMAGE position:(0.863in,8.483in) / (93px,916px) size:(2.400in,0.667in) / (259px,72px) -->
   <!-- IMAGE_END -->
   ```

   Note: The comment lines for images might not be specified in one location in the HTML file.
2. Copy the HTML comment lines in the HTML file (such as `afpdoc`*x*`.html`) 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.
3. Add image information between the starting <IMAGE> line and ending <IMAGE_END> line for the images you want transformed.

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 transformed GIF 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 **afp2web** 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 GIF 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) / (93px,916px) size:(2.400in,0.667in) / (259px,72px) -->
<!-- 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) / (567px,66px) size:(0.667in,0.800in / (72px,86px) -->
//<!-- IMAGE_END -->
<!-- IMAGE position:(0.863in,8.483in) / (93px,916px) size:(2.400in,0.667in) / (259px,72px) -->
<!-- IMAGE_END -->
<!-- IMAGE position:(3.596in,8.550in) / (388px,923px) size:(2.633in,0.700in) / (284px,76px) -->
<!-- IMAGE_END -->
<!-- IMAGE name:(S1PSEG01) position:(6.162in,8.483in) / (666px,916px) size:(2.067in,0.604in) / (223px,65px) -->
<!-- IMAGE_END -->
```

3. Run **afp2web** with the -g parameter to generate GIF image output files that contains only the image that you commented out. For example: `afp2web -a -g c:\documents\afpdoc.afp`, where -a creates separate files for each page, -g creates GIF image files, and `c:\documents\afpdoc.afp` is the directory and file name of the AFP document you are transforming.
4. View the GIF 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 GIF 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 GIF output file. Empty entries do not include any image information between the starting <IMAGE> and ending <IMAGE_END> lines, as in Figure 8.

```
<!-- IMAGE position:(0.863in,8.483in) / (93px,916px) size:(2.400in,0.667in) / (259px,72px) -->
<!-- IMAGE_END -->
```

Figure 8. Empty entries in the image map configuration file

Therefore, to remove AFP file images so they are not generated in the GIF 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 GIF file.
2. Run **afp2web** on the AFP file (for example, `afp2web -a -g c:\documents\afpdoc.afp`). A GIF file is generated (such as `c:\documents\afpdoc1.gif`) that does not contain any images for those empty entries in the configuration file.

Substituting existing images with AFP2HTML Transform

During the AFP2HTML Transform process, you can use the image map configuration file to substitute an AFP image with a previously generated image. This image can be one created with the **afp2web** command or one created from an image editing program.

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 pixels, of the left edge of the image 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 pixels, of the top edge of the image 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 pixels) reserved on the page for the image, regardless of the size of the image. The XSize value is usually set to the actual width of the image (for the best browser performance). If a value other than the actual width is supplied, the Web browser scales the image to fit the given width value. 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 pixels) reserved on the page for the image, regardless of the size of the image. The YSize value is usually set to the actual height of the image (for the best browser performance). If a value other than the actual height is supplied, the Web browser scales the image to fit the given height value. This parameter is optional because the vertical size of the incoming AFP image is used if YSize is not specified.

URL=*path*
Specifies the relative or absolute location of the image file. This text should be enclosed in double quotes if a blank is used within the URL value.

ZIndex=*n*
Sets the stacking order relative to other image definitions. If multiple images are defined for a page, the ZIndex parameter controls which image is shown for the overlapping areas. The valid values are **0** to **4**, with "0" being in the back of the stacking order and "4" being in the front. ZIndex is optional.

As shown in Figure 9 on page 23, when the first image is encountered it is substituted with "logo1.gif"; when the second image is encountered, it is substituted with "logo2.gif".

```
<!-- IMAGE position:(5.250in,0.613in) / (567px,66px) size:(0.667in,0.800in / (72px,86px) -->
IMAGE XPos=0 YPos=0 XSize=900 YSize=200 URL="http://www.abc.com/images/logo1.gif" ZIndex=1
<!-- IMAGE_END -->
<!-- IMAGE position:(0.863in,8.483in) / (93px,916px) size:(2.400in,0.667in) / (259px,72px) -->
IMAGE XPos=0 YPos=0 XSize=500 YSize=300 URL=images/logo2.gif ZIndex=2
<!-- IMAGE_END -->
<!-- IMAGE position:(3.596in,8.550in) / (388px,923px) size:(2.633in,0.700in) / (284px,76px) -->
<!-- IMAGE_END -->
<!-- IMAGE name:(S1PSEG01) position:(6.162in,8.483in) / (666px,916px) size:(2.067in,0.604in) / (223px,65px) -->
<!-- IMAGE_END -->
```

Figure 9. 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 10, 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 URL="http://www.abc.com/images/logo1.gif" ZIndex=1
<!-- IMAGE_END -->
```

Figure 10. 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 COLORED_AREA definition parameters between the starting IMAGE and ending IMAGE_END lines of an image information entry in the image map configuration file. The colored area definition parameters are:

XPos=*n*
Defines the position, in pixels, 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 pixels, 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 pixels) 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 pixels) 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.

Color=*value*

Specifies the color of the area. The color specification matches that used in HTML, where *value* is **rgb (***n,n,n***)** or a plain language equivalent, such as **blue** (refer to an HTML reference for accepted values). Examples include:

```
Color=green
Color=#00FF00
Color=rgb(0,255,0)
Color=rgb(0%,100%,0%)
```

For example, in Figure 11 when the first image is encountered, it is substituted with a red, 86x72 pel area positioned at (567,66).

```
<!-- IMAGE position:(5.250in,0.613in) / (567px,66px) size:(0.667in,0.800in / (72px,86px) -->
COLORED_AREA XPos=567 YPos=66 XSize=72 YSize=86 Color=red>
<!-- IMAGE_END -->
<!-- IMAGE position:(0.863in,8.483in) / (93px,916px) size:(2.400in,0.667in) / (259px,72px) -->
<!-- IMAGE_END -->
<!-- IMAGE position:(3.596in,8.550in) / (388px,923px) size:(2.633in,0.700in) / (284px,76px) -->
<!-- IMAGE_END -->
<!-- IMAGE name:(S1PSEG01) position:(6.162in,8.483in) / (666px,916px) size:(2.067in,0.604in) / (223px,65px) -->
<!-- IMAGE_END -->
```

Figure 11. 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, or size. If the incoming AFP image matches all the characteristics listed for the image entry, the shaded area is substituted.

For example, in Figure 12, the size values are used. When an incoming AFP image matches the image size information of 72x86 pixels, a red colored area is created. The XPos, YPos, XSize, and YSize parameters for the rectangle are extracted from the AFP image.

```
<!-- IMAGE size:(0.667in,0.800in) / (72px,86px) -->
COLORED_AREA Color=red>
<!-- IMAGE_END -->
```

Figure 12. 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. AFP2HTML Transform can include an image, which emulates the preprinted form, in the transformed output. The transform only generates a reference to the image URL. Therefore, any image supported by Web browsers can be used, such as GIF, JPEG, and PNG). The image can be in color and can be included on all the pages or only the first page created.

To include an image that emulates a preprinted form, add the STATIC_IMG definition parameters between the starting IMAGE and ending IMAGE_END lines in the image map configuration file:

These parameters are used with the static image definition:

XPos=*n*
Defines the position, in pixels, of the left edge of the image relative to the left edge of the page.

YPos=*n*
Defines the position, in pixels, of the top edge of the image relative to the top edge of the page.

XSize=*n*
Defines the width (horizontal size in pixels) reserved on the page for the image, regardless of the size of the image. The XSize value is usually set to the actual width of the image (for the best browser performance). If a value other than the actual width is supplied, the Web browser scales the image to fit the given width value.

YSize=*n*
Defines the height (vertical size in pixels) reserved on the page for the image, regardless of the size of the image. The YSize value is usually set to the actual height of the image (for the best browser performance). If a value other than the actual height is supplied, the Web browser scales the image to fit the given height value.

URL=*path*
Specifies the relative or absolute location of the image file. This text should be enclosed in double quotes if a blank is used within the URL value. Type Valid values include 'ALL' or 'FIRST' and specifies whether the image should be included on all pages or just the first output page generated.

Type=All | First
Specifies on which pages the image is included The values include:

All
The image is included on all pages.

First
The image is included on the first page only.

ZIndex=*n*
Sets the stacking order relative to other image definitions. If multiple images are defined for a page, the ZIndex parameter controls which image is shown for the overlapping areas. The valid values are **0** to **4**, with "0" being in the back of the stacking order and "4" being in the front. ZIndex is optional.

In the example in Figure 13, the GIF image, "form1.gif", is included on all pages in the HTML output.

```
<!-- IMAGE -->
STATIC_IMG XPos=0 YPos=0 XSize=72 YSize=722 URL=="http://www.abc.com/images/form1.gif" Type=ALL ZIndex=1 -->
<!-- IMAGE_END -->
<!-- IMAGE position:(0.863in,8.483in) / (93px,916px) size:(2.400in,0.667in) / (259px,72px) -->
<!-- IMAGE_END -->
<!-- IMAGE position:(3.596in,8.550in) / (388px,923px) size:(2.633in,0.700in) / (284px,76px) -->
<!-- IMAGE_EN --D>
<!-- IMAGE name:(S1PSEG01) position:(6.162in,8.483in) / (666px,916px) size:(2.067in,0.604in) / (223px,65px) -->
<!-- IMAGE_END -->
```

Figure 13. Example of an image added to HTML output

Note: The image definitions do not include name, 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.

Chapter 6. Application programming interfaces

The AFP2HTML Transform application programming interfaces (APIs) convert AFP documents and resources into files that can be displayed with a Web browser. These APIs are written to interface with a C/C++ application.

The AFP2HTML 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 AFP2HTML Transform command line interface).

The AFP2HTML 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 AFP2HTML Transform.

Packaging information

This section describes the API packaging for the AFP2HTML 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 AFP2HTML Transform as well as several subdirectories. Figure 14 on page 28 shows the file structure for APIs used by AFP2HTML Transform.

© Copyright IBM Corp. 2007, 2010

```
afp2webd.dll
ftdafp.dll
ftdbdt.dll
ftdfltc.dll
ftdodhtm.dll
ftdodwjp.dll
ftdport2.dll
ftdportb.dll
ftdres2.dll
ftdwser.dll
ftdwvsih.dll
\font
    alias.fnt
    coded.fnt
    cpdef.fnt
    csdef.fnt
    icoded.fnt
    \AFM
    \maps
```

Figure 14. File structure for AFP2HTML Transform APIs on Windows servers

UNIX server

The package consists of a shared library file (`afp2web_shr`) and a `\font` subdirectory in the directory where the shared library is located. The `/font` subdirectory contains the font definition files needed by AFP2HTML Transform as well as several subdirectories. Figure 15 shows the file structure for APIs used by AFP2HTML Transform.

```
/font
    alias.fnt
    coded.fnt
    cpdef.fnt
    csdef.fnt
    icoded.fnt
    /AFM
    /maps
```

Figure 15. File structure for AFP2HTML Transform APIs on UNIX servers

IBM i OnDemand server

The package consists of a shared library file (`afp2web_shr`) and a `/font` subdirectory in the directory where the shared library is located. The `/font` subdirectory contains the font definition files needed by AFP2HTML Transform as well as several subdirectories. Figure 16 shows the directory structure for the AFP2HTML Transform.

```
/QIBM/ProdData/OnDemand/www/binhtml
/QIBM/ProdData/OnDemand/www/binhtml/font
/QIBM/ProdData/OnDemand/www/binhtml/font/locale/uconvtab
/QIBM/ProdData/OnDemand/www/binhtml/font/maps
/QIBM/ProdData/OnDemand/www/binhtml/font/AFM
/QIBM/ProdData/OnDemand/www/binhtml/java_api
```

Figure 16. Directory structure for AFP2HTML Transform on IBM i OnDemand servers

Figure 17 shows the file structure for APIs used by AFP2HTML Transform.

```
afp2web
afp2web_shr
afp2webd.h
insure.afp
split_afp2html
/font
    alias.fnt
    coded.fnt
    cpdef.fnt
    csdef.fnt
    icoded.fnt
    /AFM
    /locale/uconvtab
        Ibm-1363
        Ibm-1386
        Ibm-300
        Ibm-834
        Ibm-835
        Ibm-837
        Ibm-943
        Ibm-950
    /maps
/java_api
    AFP2WebDemo.java
    AFP2WebServlet.java
    afp2web.pdf
    afp2web.jar
    liba2wjni.so
    testa2ws.java
    a2weip82.jar
    eClient82_howto.pdf
```

Figure 17. File structure for AFP2HTML 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 18 shows an API DLL dynamically loaded with the Windows LoadLibrary function.

```
// Load the Afp2web transform DLL.
//
HINSTANCE hXForm = LoadLibrary("c:\\afp2web\\afp2webd.dll");
if (hXForm == NULL) {
  fprintf(stderr, "Could not load the AFP2web transform DLL");
}
```

Figure 18. Example of loading a Windows API DLL

Figure 19 on page 30 shows how to dynamically obtain the function pointer for the A2WDocStart function with the Windows GetProcAddress call and then call the A2WDocStart function with the function pointer.

```
// Define a fpA2WDocStart function pointer variable for the A2WDocStart function.
//
void* (__cdecl * fpA2WDocStart)(unsigned char*, unsigned long, char*);

// Obtain the A2WDocStart function pointer / entry point in the DLL.
//
fpA2WDocStart = (void* (__cdecl *)(unsigned char*, unsigned long, char *))
                GetProcAddress(hXForm, "A2WDocStart");
if (fpA2WDocStart == NULL) {
  fprintf(stderr, "Could not load the function pointer for A2WDocStart");
}

// Call the transform function to start processing an AFP document.
//
void* hDocument = (*fpA2WDocStart)(pInBuffer, iDataLen, "c:\\afp2web\\");
```

Figure 19. 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 20 for AIX, Figure 21 for Sun and Linux, and Figure 22 on page 31 for z/OS UNIX System Services.

AIX:

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

// Load the Afp2web transform shared library which returns the main entry point.
//
fpA2WGetFuncPtr=(void* (*)(const char*)) load ("/a2wXForm/afp2web_shr", 1, NULL);
if (fpA2WGetFuncPtr == NULL) {
  fprintf(stderr, "Could not load the AFP2web transform library");
}
```

Figure 20. Example of loading an API shared library on AIX

Sun and Linux:

```
// Load the Afp2web transform shared library which returns handle to the library.
//
void* hSLib= dlopen ("/a2wXForm/afp2web_shr", RTLD_LAZY);
if (hSLib == NULL) {
  fprintf(stderr, "Could not load the AFP2web transform library");
}
```

Figure 21. Example of loading an API shared library on Sun and Linux

z/OS UNIX System Services:

```
// Load the Afp2web transform shared library which returns handle to the library.
//
dllhandle* hSLib= dllload ("/a2wXForm/afp2web_shr");
if (hSLib == NULL) {
  fprintf(stderr, "Could not load the AFP2web transform library");
}
```

Figure 22. 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 23 for AIX, Figure 24 for Sun and Linux, and Figure 25 on page 32 for z/OS UNIX System Services. The function pointer is obtained for the A2WDocStart function with the main entry point from the load call and then the A2WDocStart function is called with the function pointer.

AIX:

```
// Define a fpA2WDocStart function pointer variable for the A2WDocStart function.
//
void* (* fpA2WDocStart)(unsigned char*, unsigned long, char*);

// Obtain the A2WDocStart function pointer / entry point in the library.
//
fpA2WDocStart = (void* (*)(unsigned char*, unsigned long, char *))
                (*fpA2WGetFuncPtr)(hXForm, "A2WDocStart");
if (fpA2WDocStart == NULL) {
  fprintf(stderr, "Could not load the function pointer for A2WDocStart");
}

// Call the transform function to start processing an AFP document.
//
void* hDocument = (*fpA2WDocStart) (pInBuffer, iBytesRead, "/a2wXForm/");
```

Figure 23. Example of obtaining the function pointer to API options on AIX

Sun or Linux:

```
// Define a fpA2WDocStart function pointer variable for the A2WDocStart function.
//
void* (* fpA2WDocStart)(unsigned char*, unsigned long, char*);

// Obtain the A2WDocStart function pointer / entry point in the library.
//
fpA2WDocStart = (void* (*)(unsigned char*, unsigned long, char *))
                dlsym(hSLib, "A2WDocStart");
if (fpA2WDocStart == NULL) {
  fprintf(stderr, "Could not load the function pointer for A2WDocStart");
}

// Call the transform function to start processing an AFP document.
//
void* hDocument = (*fpA2WDocStart) (pInBuffer, iBytesRead, "/a2wXForm/");
```

Figure 24. Example of obtaining the function pointer to API options on Sun or Linux

z/OS UNIX System Services:

```
// Define a fpA2WDocStart function pointer variable for the A2WDocStart function.
//
void* (* fpA2WDocStart)(unsigned char*, unsigned long, char*);

// Obtain the A2WDocStart function pointer / entry point in the library.
//
fpA2WDocStart = (void* (*)(unsigned char*, unsigned long, char *))
                dllqueryfn (hSLib, "A2WDocStart");
if (fpA2WDocStart == NULL) {
  fprintf(stderr, "Could not load the function pointer for A2WDocStart");
}

// Call the transform function to start processing an AFP document.
//
void* hDocument = (*fpA2WDocStart) (pInBuffer, iBytesRead, "/a2wXForm/");
```

Figure 25. Example of obtaining the function pointer to API options on z/OS UNIX System Services

IBM i OnDemand server

Figure 26 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 Afp2web transform shared library which returns the main entry point.
//
fpA2WGetFuncPtr=(void* (*)(const char*))
  load ("/QIBM/UserData/OnDemand/www/binhtml/afp2web_shr", 1, NULL);
if (fpA2WGetFuncPtr == NULL) {
  fprintf(stderr, "Could not load the AFP2web transform library");
}
```

Figure 26. Example of loading an API shared library on an IBM i OnDemand server

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

AFP2HTML Transform API

The AFP2HTML Transform API converts AFP documents and resources into data that can be distributed over the Internet and displayed with a Web browser. This API was created so that applications, such as a common gateway interface (CGI), can transform data without creating a new process, and can reduce some of the overhead of data I/O.

The AFP2HTML Transform API generates data that can be sent directly to a Web browser and tries to replicate the printed output. The text and lines in the AFP data are transformed into HTML tags. Images and graphics in the AFP data can be ignored, substituted with an existing Web browser image file (such as GIF or JPEG), or dynamically transformed into a Web browser image.

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 option 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 `afp2webd.h` and needs to be included by the calling application. On the Windows operating system, the calling application should compile so the "structure member alignment" is on byte boundaries. On all other operating systems, the calling application should use the default "structure member alignment".

The input options structure is referenced with the name, `A2WCvtOpts`. The definition of this structure is shown in Figure 27

```
typedef struct
{
  int  fCreateGIF;                    // Create GIF file for image data
  int  fShadeFlag;                    // Create shaded areas for all images
  int  fSuppressFonts;                // Suppress fonts listed in 'imagfont.cfg'
  int  iPageNumber;                   // Page number to convert
  int  iRotation;                     // Rotation setting (0 | 90 | 180 | 270)
  int*  piHtmlOutLen;                 // Returned HTML output buffer length
  int*  piImgOutLen;                  // Returned image output buffer length
  char**  ppHtmlOutBuf;               // Returned HTML output buffer pointer
  unsigned char**  ppImgOutBuf;       // Returned image output buffer pointer
  char  szDefShadeColor[256];         // Default shade color (used with fShadeFlag)
  char  szFontMapFile[SMAX_PATH+1];   // Fully qualified 'imagfont.cfg' file spec
  char  szOptionsFile[SMAX_PATH+1];   // Fully qualified transform options file spec
  char  szImageMapFile[SMAX_PATH+1];  // Fully qualified 'imagemap.cfg' file spec
  char  szImageRef[SMAX_PATH+1];      // Image reference name placed in HTML
  char  szScale[256];                 // Scale factor (ex. 1.50) - float number string
  int   iVersion;                       // Structure version number
  char  szFormDef[SMAX_PATH+1];         // Fully qualified form definition file
  char  szProfile[SMAX_PATH+1];         // Fully qualified transform options file
} A2WCvtOpts;
```

Figure 27. Input options structure

The options in the A2WCvtOpts structure are:

fCreateGIF
Generates a single GIF image file for the AFP image and graphic data on a page. If the image map configuration file is used, and images on the page are defined in the configuration file, these images are not included as part of the GIF output image. See Chapter 5, "Mapping AFP images," on page 19 for more information about using the configuration file to identify images.

If this option is used, the szImageRef options might need to also be specified to indicate the GIF image reference file name that is embedded inside the associated HTML output file. This option is typically used in coordination with the piImgOutLen and the ppImgOutBuf options, which specify the length and pointer to the image output buffer, and is returned by the transform API.

fShadeFlag
Used as a special-purpose flag. Normally set to "0" for false.

fSuppressFonts
Suppresses the display of fonts specified in the font map configuration file when the flag is set to "1" for true. The font map configuration file is named `imagfont.cfg` by default. It is located in the same directory where the transform API code was installed. You can change the name and location of this file with the szFontMapFile option.

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 is specified (for example, out of range), the first page in the document is converted.

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.

piHtmlOutLen
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 HTML data. The output HTML data buffer is returned in the ppHtmlOutBuf option. The calling application should check the value of the variable after the conversion process. A "0" value indicates an error has occurred and any data in the output HTML data buffer is not valid.

piImgOutLen
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 image data. The output image data buffer is returned in the ppImgOutBuf option. The calling application should check the value of the variable after the conversion process. A "0" value indicates an error has occurred and any data in the output image data buffer is not valid.

ppHtmlOutBuf
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 HTML data. The length of the output HTML data buffer is returned in the piHtmlOutLen option. The calling application should check the value of the variable after the conversion process. A null value indicates an error has occurred.

ppImgOutBuf
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 image data. The length of the output image data buffer is returned in the piImgOutLen option. The calling application should check the value of the variable after the conversion process. A null value indicates an error has occurred.

szDefShadeColor
Used as a special purpose variable. Normally set to an empty string.

szFontMapFile
Specifies a fully-qualified path and file name string for the font map configuration file. The configuration file lists fonts that require special processing as well as other processing options. See "Font map configuration file" on page 17 for more information about using the configuration file.

The font map configuration file is named `imagfont.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.

If the font map configuration file is available and the fSuppressFonts option is specified, the text using any of the fonts listed in the configuration file is not displayed.

For Windows servers only: If the font map configuration file is available (`imagfont.cfg`), the fSuppressFonts option is not specified, and the fCreateGIF option is specified, the text using any of the fonts listed in the configuration file is included as part of the output GIF image.

szOptionsFile
Specifies a fully-qualified path and file name string for the AFP2HTML Transform options 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. See Chapter 5, "Mapping AFP images," on page 19 for more information about identifying images using the configuration file.

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.

szImageRef
Specifies the HTML tag that references the GIF file to be displayed in the HTML output. This option is used when the fCreateGIF flag is turned on and specifies the location of the GIF image file relative to the associated HTML file.

The file name of the GIF image, along with any other URL information (either absolute or relative), can be specified. For example, if the image file resides in the same location (either on the client or the server) as the HTML output, specify just the image file name, such as `DocAPage1.gif`.

If the image file resides in a subdirectory relative to the associated HTML output (either on the client or the server), a directory and file name can be given, such as `docimgs/DocAPage1.gif`. If the image resides in a specific location on the server, an absolute URL string can be specified, such as `http://www.ibm.com/docimgs/DocAPage1.gif`.

szScale
Specifies a scale factor for the output as a floating point number string. The default value is **1.0**. Specifying a string value of "2.0" scales the output to twice as large as the default value; specifying a string value of "0.5" scales the output to one half of the default value.

iVersion
Version number of the A2WCvtOpts structure. This number must be **3**.

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

szProfile
Specifies a fully-qualified path and file name string for the transform options file, which is named `afp2web.ini` by default.

Available programming functions

This section contains information about the programming functions available for the AFP2HTML Transform API.

A2WDocStart2

A2WDocStart2 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 correct document being processed. A document handle of NULL indicates that an error has occurred.

The format of A2WDocStart2 is:

```
void* A2WDocStart2 (unsigned char* pDataIn, unsigned long ulSizeIn, char* pszDir, A2WCvtOpts* pCvtOpts)
```

The options in the A2WDocStart2 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 AFP2HTML 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.

A2WGetPageCount

A2WGetPageCount returns the number of pages in a specific AFP document.

The format of A2WGetPageCount is:

```
int A2WGetPageCount (void* hAfpDoc)
```

The option in the A2WGetPageCount function is:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2WDocStart 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 option specified is not valid.

A2WXFormPage

A2WXFormDoc transforms a page in an AFP document using the given values in the input options structure.

The format of A2WxFormPage is:

```
void* A2WXFormPage (void* hAfpDoc, A2WCvtOpts* pCvtOpts)
```

The options in the A2WFormPage function are:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2WDocStart 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 HTML output buffer values specified in the pCvtOpts structure are not valid.

-40 Image output buffer values specified in the pCvtOpts structure are not valid.

A2WXFormDoc

A2WXFormDoc transforms the entire AFP document using the given values in the input options structure. The pages of the document are concatenated into a single HTML output object. This function should not be used on documents more than a few pages in length otherwise the resultant HTML can become quite large. Users trying to display these large HTML documents might experience poor response time within the Web browser application.

The format of A2WxFormDoc is:

```
void* A2WXFormDoc (void* hAfpDoc, A2WCvtOpts* pCvtOpts)
```

The options in the A2WFormDoc function are:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2WDocStart 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 HTML output buffer values specified in the pCvtOpts structure are not valid.

Special consideration must be taken if GIF images are to be also generated by the transform. The image output buffer values specified in the pCvtOpts structure are not used. See “A2WGetGifArray” on page 38, “A2WClearGIFArray” on page 40, or “Processing GIF files” for more details about generating GIF images using this function.

Processing GIF files: When using the A2WXFormDoc function with the AFP2HTML API, special consideration must be taken if GIF images are to be generated by the transform. The image output buffer values available in the A2WCvtOpts structure are not used because multiple GIF images might be generated on a page by page basis but the A2WCvtOpts structure interface only allows the return of a single image.

To generate GIF images, a configuration file named `ImageFileOut.cfg` is used. The transform places each GIF image in a randomly generated named file in the directory defined in the configuration file. The HTML output is modified to reference the associated files as needed.

The `ImageFileOut.cfg` file consists of two lines:

First line
Specifies the absolute directory where the transform saves the GIF files. The Web server must have permission to access the contents of this directory. For example, the transform puts the GIF output files in this directory on the system: `c:\inetpub\wwwroot\afp2web\images`.

Second line
Specifies the Web server's URL specification of the directory on the first line. For example, the HTML used this URL when referencing any associated GIF files: `http://www.ibm.com/afp2web/images`.

The `ImageFileOut.cfg` file must be placed in the same directory as the transform modules on the Windows operating system. For UNIX environments, this configuration file must be place in the current working directory when the transform function is called.

A2WDocEnd

A2WDocEnd ends the processing for a given AFP document.

The format of A2WDocEnd is:

```
void* A2WDocEnd (void* hAfpDoc, A2WCvtOpts* pCvtOpts)
```

The options in the A2WFormDoc function are:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2WDocStart 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.

A2WGetGifArray

A2WGetGIFArray provides the mechanism to return the GIF images generated during the transform process when used with the A2WXFormDoc function. When converting the entire document, the HTML output is returned in a single object whereas GIF images are generated on a page by page basis. This function should be called after A2WXFormDoc and before A2WDocEnd and returns all of the associated GIF images for the entire document in an array structure.

Note: This function is only available on special builds.

The format of A2WGetGIFArray is:

```
int A2WGetGIFArray (void* hAfpDoc, void** ppGIFArray, int* piGIFArrayCnt)
```

The options in the A2WGetGIFArray function are:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2WDocStart function.

ppGIFArray
Specifies the address of the variable the calling program allocates as a "void *" type. The function modifies the reference value to point to the GIF array if any GIF images are generated for the document.

piGIFArrayCnt
Specifies the address of the variable the calling program allocates as an "int" type. The function modifies the reference value to the number of GIF images that are generated for the document and are available in the associated GIF array.

The ppGIFArray and piGIFArrayCnt options are passed by reference and let information be returned back to the caller. The value returned in the piGIFArrayCnt variable should be greater than zero before trying to read the data referred to by the ppGIFArray variable.

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.

-20 Null value specified for the ppGIFArray or piGIFArrayCnt option.

The GIF array is a collection of entries consisting of this structure:

```
struct A2HGIFOut {
     unsigned char * pucGIFContents; // GIF image
     unsigned long ulGIFLength; // length of GIF in bytes
};
```

This structure is defined in the header file `afp2webd2.h` and needs to be included by the calling application. On the Windows operating system, the calling application should compile so the "structure member alignment" is on byte boundaries. On all other operating systems, the calling application should use the default "structure member alignment" when compiling.

If the transform generates one or more GIF images, this function returns a reference to the array that was generated. These lines demonstrate how this function might be called as well as stepping through the GIF array to obtain the individual GIF images:

```
// Allocate the variables to get the GIF array
//
void* pGifArray = 0;
int iGifCnt = 0;
A2HGIFOut* pGifArrayEntry;

// Call the function to obtain the GIF array
//
iRetCode = A2WGetGIFArray ( hDocument, &pGifArray, &iGifCnt );

// Process the GIF array if the transform generated entries
//
if ( iGifCnt > 0 ) {

     // The returned GIF array reference will point to the first
        entry
     //
     pGifArrayEntry = (A2HGIFOut*) pGifArray;

    // Loop through all of the images in the array
    //
    for (int idx = 0; idx < iGifCnt; idx++) {
           Imagedata = pGifArrayEntry[idx].pucGIFContents;
           ImageLength = pGifArrayEntry[idx].ulGIFLength;
    }
}
```

To associate the transform created images with the GIF image references inside the HTML, a simple numbering scheme is used. When used with the A2WXFormDoc function, the szImageRef entry in the A2WCvtOpts structure contains the URL reference base name. This special version of the code concatenates a sequence number to the end of the specified szImageRef string. For example, if the szImageRef string is `http://www.abc.com/getimages.jsp?ID=2&NUMBER=`, "1" is added to the end of the string for the first image generated, "2" is added to the end of the string for the second image generated, and so on.

If the `ImageFileOut.cfg` configuration file happens to also be in use (see "Processing GIF files" on page 37), the configuration file processing takes precedence over this function. Any GIF images created by the transform is output to files as specified by the `ImageFileOut.cfg` configuration. None of the GIF images is output to this function's GIF array.

A2WClearGIFArray

A2WClearGIFArray releases any storage allocated by the transform for the GIF array. The GIF array is generated when the transform creates GIF images while performing A2WXFormDoc function. This function should be called before A2WDocEnd otherwise memory leaks could result.

Note: This function is only available on special builds.

The format of A2WClearGIFArray is:

```
int A2WClearGIFArray (void* hAfpDoc)
```

The option in the A2WClearGIFArray function is:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2WDocStart function.

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.

A2WGenerateMessages

A2WGenerateMessages 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.

Note: This function is only available on special builds.

The format of A2WGenerateMessages is:

```
int A2WGenerateMessages (void* hAfpDoc, int iSwitch)
```

The options in the A2WGenerateMessages function are:

hAfpDoc
Identifies the AFP document object handle that is returned from the A2WDocStart 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.

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.

© Copyright IBM Corp. 2007, 2010

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample

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

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.

AFP. See Advanced Function Presentation.

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.

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.

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

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

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),

© Copyright IBM Corp. 2007, 2010

contains text or graphics (data-file object), or processes a series of commands (batch file).

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 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

GIF. See Graphics Interchange Format.

GOCA. See Graphics Object Content Architecture.

Graphics Interchange Format (GIF). A file format for storing images. GIF files are common on the World Wide Web because they only contain a maximum of 256 colors and are therefore very small.

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.

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.

J

Joint Photographic Experts Group (JPEG). The standard for the compression of digitized continuous-tone images.

JPEG. See Joint Photographic Experts Group.

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.

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.

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.

pel. See picture element.

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.

pixel. See picture element.

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.

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

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

A

A2WClearGIFArray 40
A2WDocEnd 38
A2WDocStart2 35
A2WGenerateMessages 41
A2WGetGIFArray 38
A2WGetPageCount 36
A2WXFormDoc 37
A2WXFormPage 36
AFP resources, using 9
AFP2HTML
  afp2web command 5
  API input options structure 32
  application programming interfaces for 27
  benefits of 1
  configuring 5
  font files supplied 11
  font map configuration file 17
  GIF image configuration file 37
  images, mapping AFP 19
  installing 3
  limitations 1
  mapping fonts 11
  options file 7
  overview of 1
  profile 8
  requirements 3
afp2web command 5
alias file 17
APIs
  See application programming interfaces
application programming interfaces
  A2WClearGIFArray 40
  A2WDocEnd 38
  A2WDocStart2 35
  A2WGenerateMessages 41
  A2WGetGIFArray 38
  A2WGetPageCount 36
  A2WXFormDoc 37
  A2WXFormPage 36
  input options structure 32
  loading code and obtaining function pointers 29
  packaging information 27
  processing GIF files with A2WXFormDoc 37
  programming functions 35

B

benefits 1

C

character set definition file 12
code loading for APIs 29
code page map file 16
coded font file 12
coded page definition file 15
coded.fnt 12
colored areas, substituting shaded images with 23
command, afp2web 5
configuration files
  font map 17
  GIF image 37
  image map
    creating 19
    defining colored area images with 23
    identifying AFP images in 21
    including preprinted form images with 24
    removing images with 21
    substituting existing images with 22
configuring AFP2HTML 5
cpdef.fnt 15
csdef.fnt 12

E

empty entries 21

F

files for mapping fonts 11
font files
  alias.fnt 17
  code page map file 16
  coded.fnt 12
  cpdef.fnt 15
  csdef.fnt 12
  icoded.fnt 12
  supplied with AFP2HTML 11
font map configuration file 17
fonts
  files for mapping 11
  mapping AFP 11
function pointers for APIs 29

G

GIF files, processing with A2WXFormDoc 37
GIF image configuration file 37

I

IBM i server
  installing AFP2PDF on 3
  loading code for APIs 32
  packaging for APIs 28
icoded.fnt 12
image map configuration file
  creating 19
  defining colored area images with 23
image map configuration file *(continued)*
  empty entries 21
  identifying AFP images in 21
  including preprinted form images with 24
  removing images with 21
  substituting existing images with 22
ImageFileOut.cfg file 37
imagemap.cfg 19
images
  configuration file for 19
  defining colored areas with configuration file 23
  empty entries 21
  identifying in configuration file 21
  including preprinted forms with configuration file 24
  mapping AFP 19
  removing with configuration file 21
  substituting existing with configuration file 22
input options structure for API 32
installing AFP2HTML 3

L

limitations 1
loading code for APIs 29

M

mapping
  AFP fonts 11
  AFP images 19
  files for fonts 11
  images with configuration file 19

O

OnDemand, installing AFP2HTML with 4
options file 7
options, API input 32
overview 1

P

packaging information for APIs 27
parameters
  AFP2HTML Transform profile 8
  afp2web command 5
  font map configuration file 17
  options file 7
preprinted forms, including images from 24
profile, AFP2HTML Transform 8
programming functions available for API 35

© Copyright IBM Corp. 2007, 2010

R

requirements 3
resources, using AFP 9
return codes for afp2web command 7

S

shaded images, substituting colored areas for 23
structure for API input options 32
syntax for afp2web command 5

T

transform options file 7

U

UNIX server
  loading shared libraries for APIs 30
  obtaining function pointers for APIs 31
  packaging for APIs 28

W

Windows server
  loading code and obtaining function pointers for APIs 29
  packaging for APIs 27

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

SC19-2945-00